



D O M A I N

Order No. 005798
Revision 01

***DOMAIN/IX Command Reference
for System V***



**DOMAIN/IX Command Reference
for System V**

**Order No. 005798
Revision 01**

**Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824**

Apollo Computer Inc. reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Apollo Computer Inc. to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF APOLLO COMPUTER INC. HARDWARE PRODUCTS AND THE LICENSING OF APOLLO COMPUTER INC. SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN APOLLO COMPUTER INC. AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY APOLLO COMPUTER INC. FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY BY APOLLO COMPUTER INC. WHATSOEVER.

IN NO EVENT SHALL APOLLO COMPUTER INC. BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATING TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF APOLLO COMPUTER INC. HAS BEEN ADVISED, KNEW, OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

THE SOFTWARE PROGRAMS DESCRIBED IN THIS DOCUMENT ARE CONFIDENTIAL INFORMATION AND PROPRIETARY PRODUCTS OF APOLLO COMPUTER INC. OR ITS LICENSORS.

THIS SOFTWARE AND DOCUMENTATION ARE BASED IN PART ON THE FOURTH BERKELEY SOFTWARE DISTRIBUTION UNDER LICENSE FROM THE REGENTS OF THE UNIVERSITY OF CALIFORNIA.

© 1986, 1987 Apollo Computer Inc. All rights reserved.

Printed in U.S.A.

First Printing: December 1986

This document was formatted on a DOMAIN System using the troff text formatter distributed with DOMAIN[®]/IX software.

APOLLO and DOMAIN are registered trademarks of Apollo Computer Inc.

AEGIS, DGR, DOMAIN/Bridge, DOMAIN/DFL-100, DOMAIN/DQC-100, DOMAIN/Dialogue, DOMAIN/IX, DOMAIN/Laser-26, DOMAIN/PCI, DOMAIN/SNA, D3M, DPSS, DSEE, EtherBridge, GMR, and GPR are trademarks of Apollo Computer Inc.

DEC, PDP, and VAX are trademarks of Digital Equipment Corporation.

TEKTRONIX is a registered trademarks of Tektronix, Inc.

HP is a trademark of Hewlett-Packard, Inc.

PREFACE

The DOMAIN[®]/IX[™] *Command Reference Manual for System V* consists of user commands and games that run on DOMAIN/IX or other implementations of the UNIX[®] Operating System.

Audience

This *Command Reference Manual* is intended for users who are familiar with System V UNIX software and DOMAIN networks. We recommend that you read one of the following tutorial introductions if you are not already familiar with the UNIX operating system.

- Bourne, Stephen W. *The UNIX System*. Reading: Addison-Wesley, 1982.
- Kernighan, Brian W. and Rob Pike. *The UNIX Programming Environment*, Englewood Cliffs, Prentice-Hall, 1984.
- Thomas, Rebecca and Jean Yates. *A User Guide to the UNIX System*. Berkeley: Osborne/McGraw-Hill, 1982.

This document also assumes a basic familiarity with the DOMAIN/IX system. The best introduction to the DOMAIN/IX system is *Getting Started With Your DOMAIN/IX System* (Order No. 008017). This manual explains how to use the keyboard and display, read and edit text, and manipulate files. It also shows how to request DOMAIN system services using interactive commands.

The Structure of This Document

This manual includes the following sections.

Section 1 provides reference material on user commands.

Section 6 provides reference material on games.

Sections 2 (system calls), 3 (library functions), 4 (devices files), 5 (file formats), and 7 (miscellaneous) are in the *DOMAIN/IX Programmer's Reference Manual for System V*. Section 8 (administrative commands) is in the *DOMAIN/IX Administrator's Reference Manual for System V*.

UNIX is a Registered Trademark of AT&T.

Preface

Related Volumes

The *DOMAIN/IX User's Guide* (Order No. 005803, revision 01) is the first volume you should read. It explains how DOMAIN/IX works, and contains extensive material on the Bourne Shell, C Shell, and Mail.

The *DOMAIN/IX Text Processing Guide* (Order No. 005803) describes the UNIX text editors (ed, ex, and vi) supported by DOMAIN/IX. It also contains material on the formatters troff and nroff, the macro packages ms, me, and mm, and the preprocessors eqn and tbl.

The *DOMAIN/IX Support Tools Guide* (Order No. 009413) describes various DOMAIN/IX utilities (e.g. awk(1), lex(1), yacc(1), etc.) that can help with development and maintenance of programs.

The *DOMAIN/IX Command Reference for System V* (Order No. 005798, revision 01) describes all the UNIX System V shell commands supported by the sys5 version of DOMAIN/IX.

The *DOMAIN/IX Programmer's Reference for System V* (Order No. 005799, revision 01) describes all the UNIX System V system calls and library functions supported by the sys5 version of DOMAIN/IX.

The *DOMAIN/IX Administrator's Reference for System V* (Order No. 009356) describes all the UNIX System V system administrator commands and provides detailed information on system registries and servers supported by the sys5 version of DOMAIN/IX.

The *DOMAIN/IX Programmer's Reference for BSD4.2* (Order No. 005801, revision 01) describes all the BSD4.2 UNIX system calls and library functions supported by the bsd4.2 version of DOMAIN/IX.

The *DOMAIN/IX Administrator's Reference for BSD4.2* (Order No. 009355) describes all the UNIX System V system administrator commands and provides detailed information on system registries and servers supported by the bsd4.2 version of DOMAIN/IX.

The *DOMAIN C Language Reference* (Order No. 002093) describes C program development on the DOMAIN system. It lists the features of C, describes the C library, and gives information about compiling, binding, and executing C programs.

The *DOMAIN System Command Reference* (Order No. 002547) gives information about using the DOMAIN system and describes the DOMAIN commands found in the /com directory.

The two-volume *DOMAIN System Call Reference* (Volume I Order No. 007196 revision 01, Volume II Order No. 007194 revision 01) describes calls to operating system

intro – introduction to commands	1-1
admin – create and administer SCCS files	1-2
ar – archive and library maintainer for portable archives	1-6
arcv – convert archive files from PDP-11 to common archive format	1-8
asa – interpret ASA carriage control characters	1-9
at, batch – execute commands at a later time	1-10
awk – pattern scanning and processing language	1-13
banner – make posters	1-16
basename, dirname – deliver portions of pathnames	1-17
bc – arbitrary-precision arithmetic language	1-18
bdiff – big diff	1-21
bfs – big file scanner	1-22
bs – a compiler/interpreter for modest-sized programs	1-26
cal – print calendar	1-36
calendar – reminder service	1-37
cat – concatenate and print files	1-38
cb – C program beautifier	1-39
cc – C compiler	1-40
cd – change working directory	1-43
cdc – change the delta commentary of an SCCS delta	1-44
cflow – generate C flow graph	1-46
chmod – change mode	1-49
chown, chgrp – change owner or group	1-51
cmp – compare two files	1-52
col – filter reverse line-feeds	1-53
comb – combine SCCS deltas	1-55
comm – select or reject lines common to two sorted files	1-57
cp – copy files	1-58
cpio – copy file archives in and out	1-59
cpp – the C language preprocessor	1-61
crontab – make a user crontab file	1-64
csh – a shell (command interpreter) with C-like syntax	1-67
csplit – context split	1-93
ctrace – C program debugger	1-95
cu – call another UNIX system	1-100
cut – cut out selected fields from each line of a file	1-104
cw, checkcw – prepare constant-width text for troff	1-106
cxref – generate a C program cross-reference	1-110
date – print the date	1-112
dbx – debugger	1-114
dc – desk calculator	1-122
dd – convert and copy a file	1-125
delta – make a delta (change) to an SCCS file	1-127
deroff – remove nroff/troff, tbl, and eqn constructs	1-130

diff – differential file comparator	1-132
diff3 – 3-way differential file comparison.....	1-134
diffmk – mark differences between files.....	1-136
dircmp – directory comparison	1-137
du – summarize disk usage.....	1-138
echo – echo arguments	1-139
ed, red – text editor	1-140
edit – text editor (a variant of ex for casual users)	1-150
enable, disable – enable or disable LP printers	1-151
env – set environment for command execution	1-152
eqn, neqn, checkeq – format mathematical text for nroff or troff	1-153
ex – text editor.....	1-155
expr – evaluate arguments as an expression.....	1-157
factor – factor a number	1-159
file – determine file type	1-160
find – find files	1-161
fsplit – split FORTRAN or ratfor files	1-163
gdev – graphical device routines and filters.....	1-164
get – get a version of an SCCS file.....	1-165
getopt – parse command options.....	1-172
graph – draw a graph	1-174
graphics – access graphical and numerical commands.....	1-176
grep, egrep, fgrep – search a file for a pattern.....	1-177
gutil – graphical utilities	1-179
help – ask for SCCS help	1-181
hyphen – find hyphenated words	1-182
id – print user and group IDs and names	1-183
iperm – remove a message queue, semaphore set, or shared memory ID	1-184
ipcs – report interprocess communication facilities status.....	1-185
join – relational database operator.....	1-189
kill – terminate a process	1-191
ld – link editor	1-192
lex – generate programs for simple lexical tasks.....	1-194
line – read one line.....	1-197
lint – a C program checker	1-198
ln – create a hard or soft link	1-202
login – sign on.....	1-204
logname – get log-in name	1-206
lorder – find ordering relation for an object library.....	1-207
lp, cancel – send/cancel requests to an lp line printer	1-208
lpstat – print lp status information.....	1-210
ls – list contents of directory.....	1-212
m4 – macro processor	1-215
apollo, pdp11, u3b, u3b5, vax – provide truth value about your processor type	1-219

mail, rmail – send mail to users or read mail	1-220
mailx – interactive message processing system	1-223
make – maintain, update, and regenerate groups of programs	1-237
man – print entries in this manual	1-247
mesg – permit or deny messages	1-249
mkdir – make a directory	1-250
mm – print out documents formatted with the MM macros	1-251
mmt, mvt – typeset documents, viewgraphs, and slides	1-253
mv – move files	1-255
newform – change the format of a text file	1-256
newgrp – log in to a new group	1-259
news – print news items	1-261
nice – run a command at low priority	1-262
nl – line numbering filter	1-263
nm – print name list	1-265
nohup – run a command immune to hangups and quits	1-266
nroff – format text	1-268
od – octal dump	1-270
pack, pcat, unpack – compress and expand files	1-271
passwd – change log-in password	1-274
paste – merge same lines of several files or subsequent lines of one file	1-275
pg – file perusal filter for soft-copy terminals	1-277
pr – print files	1-281
prs – print an SCCS file .	
ps – report process status	1-287
ptx – permuted index	1-290
pwd – working directory name	1-292
ratfor – rational FORTRAN dialect	1-293
regcmp – regular expression compile	1-295
rm, rmdir – remove files or directories	1-296
rmDEL – remove a delta from an SCCS file	1-297
sact – print current SCCS file editing activity	1-298
sccsdiff – compare two versions of an SCCS file	1-299
sdiff – side-by-side difference program	1-300
sed – stream editor	1-302
sh, rsh – the standard/restricted Bourne Shell (command programming language)	1-306
size – print sizes of object files	1-319
sleep – suspend execution for an interval	1-320
sort – sort and/or merge files	1-321
spell, hashmake, spellin, hashcheck – find spelling errors	1-325
spline – interpolate smooth curve	1-327
split – split a file into pieces	1-328
start_csh – start a login C Shell	1-329
start_sh – run a log-in Bourne Shell	1-330

stat – statistical network useful with graphical commands	1-331
strip – strip symbol and line number information from an object file	1-335
stty – set the options for a terminal	1-336
su – become super-user or another user	1-340
sum – print checksum and block count of a file	1-342
sync – forces write to disk	1-343
tabs – set tabs on a terminal	1-344
tail – deliver the last part of a file	1-347
tar – tape (and general purpose) archiver	1-348
tbl – format tables for nroff or troff	1-352
tee – pipe fitting	1-355
test – condition evaluation command	1-356
time – time a command	1-358
toc – graphical table of contents routines	1-359
touch – update access and modification times of a file	1-361
tplot – graphics filters	1-362
tput – query terminfo database	1-363
tr – translate characters	1-364
troff – typeset text	1-366
true, false – provide truth values	1-368
tsort – topological sort	1-369
tty – get the name of the terminal	1-370
umask – set file-creation mode mask	1-371
uname – print name of current UNIX system	1-372
unget – undo a previous get of an SCCS file	1-373
uniq – report repeated lines in a file	1-374
units – conversion program	1-375
uucp, uulog, uuname – UNIX system to UNIX system copy	1-376
uustat – uucp status inquiry and job control	1-379
uuto, uupick – public UNIX-to-UNIX system file copy	1-382
uux – UNIX-to-UNIX system command execution	1-384
val – validate SCCS file	1-387
vc – version control	1-389
vi – full screen display editor based on ex	1-392
wait – await completion of process	1-400
wc – word count	1-401
what – identify SCCS files	1-402
who – who is on the system	1-404
write – write to another user	1-406
xargs – construct argument list(s) and execute a command	1-407
yacc – yet another compiler-compiler	1-410

Preface

components that are accessible to user programs.

Documentation Conventions

Unless otherwise noted in the text, this manual uses the following symbolic conventions.

- bold** We use **bold** type to emphasize keywords in text and command-line examples. A keyword can be any of:
- The name of an executable system object (command or shell script) and any options (switches, regular expressions, or real pathnames) that command or shell script accepts. For example, **ls -la**, or **man ls**.
 - The name of a callable function, including all syntactically required punctuation. For example, **open** (*path, flags, mode*).
 - Any system object that has its own reference manual entry. For example, **passwd**(4).

We do not use bold type for general emphasis. In our ASCII help files, bold type looks the same as Roman type.

Italics We use *Italics* to emphasize:

- Names or pathnames of system objects. For example */etc/passwd* or */tmp*.
- Names we use as stand-ins for names and/or values that you must supply. For example, **man** *foo*, "...prints *filename* on standard output...", **open** (*path, flags, mode*). An example command line like

ls [*options*] [*file(s)*]

indicates that **ls** is a keyword that can be followed with one or more *options* and an optional *file* or *files*.

By extension, this font usage appears in command options and option arguments:

-n *number* Number of times to do this function

as well as in function arguments

#include *<sys/file.h>*

open(*path, flags, mode*)
char **path*;
int *flags, mode*;

We also use italics to indicate the title of a publication, such as the *DOMAIN/IX Command Reference Manual*. We do not use Italic type for general emphasis. In our ASCII help files, Italic type is underlined.

Preface

- pica** Where possible, we use the constant-width pica font (or another “type-writer” style font) in code fragments, shell or DM scripts, and scripts for commands like `awk(1)` and `sed(1)`. In our ASCII help files, pica type looks the same as Roman type.
- name(1)** Where a filename or command name is followed by a number or number-letter pair in parentheses, that number indicates the section (and, if a letter is included, the subsection) of the reference manual set in which you can find reference information on the named command or file. For example, you can find reference information on the `lex(1)` command in Section 1 of the *DOMAIN/IX Command Reference Manual* and information on the `/etc/passwd(4)` file in Section 4 of the *DOMAIN/IX Programmer's Reference Manual*.
- [brackets]** We use brackets to delimit optional command line switches (options) and arguments. Brackets are also shell metacharacters that delimit a range or character class.
- <KEY>** We enclose the name of a keyboard key in brackets. For example, `<ESC>` or `<AGAIN>`. The `<` and `>` symbols are also shell metacharacters used for redirection of input or output.
- ↑<KEY>** A control function that you execute by pressing the `<CTRL>` key and the named `<KEY>` at the same time. For example, `↑<D>` sends an End-Of-File.
- <CTRL><KEY>** Same as `↑<KEY>`.
- ...** Horizontal ellipses indicate that the preceding item can be repeated an arbitrary number of times. For example
- `troff file ...`
- means that you can say
- `troff file1 file2 file3`
- and so on.
- .
- .
- .
- We use vertical ellipses to indicate that an irrelevant portion of text has been omitted from an example.

Note that, when we begin a sentence with the name of a filesystem object, we always capitalize the first letter of the name unless this would result in an ambiguity.

Preface

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. In order to make it easy for you to communicate with us, we provide the User Change Request (UCR) system for software-related comments, and the Reader's Response form for documentation comments. By using these formal channels, you make it easy for us to respond to your comments.

You can get more information about how to submit a UCR by consulting the *DOMAIN System Command Reference*. Refer to the CRUCR (Create User Change Request) command. You can also get more information by typing:

`/com/help crucr`

in any UNIX or AEGIS shell. There is a Reader's Response form at the back of this manual. We'd appreciate it if you would take the time to fill it out when you're ready to comment on this document.

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible commands for general utility. In addition, some special commands for communication purposes are described. All commands are listed in alphabetic order, and each is suffixed by “(1)” to help identify its place in the *DOMAIN/IX Command Reference for System V*.

N.B.: Commands that relate to system maintenance, distinguished by (1M) in earlier UNIX System documentation, are described in the *DOMAIN/IX Administrator's Reference for System V*.

DIAGNOSTICS

Upon termination, each command returns two bytes of status; one is supplied by the system giving the cause for termination, and (in the case of normal termination) one supplied by the program. The former byte is 0 for normal termination; the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called exit code, exit status, or return code, and is described only where special conventions are involved.

NAME

admin – create and administer SCCS files

USAGE

admin [*options*] *files*

DESCRIPTION

Admin creates new SCCS files and changes parameters of existing ones. You may specify several different options (in any order) along with named *files*. An *s*, followed by a period, must prefix all SCCS filenames (e.g., *s.file1*).

If a named *file* does not exist, **admin** creates it. It then initializes the file's parameters according to specified options. Parameters not initialized by a valid option discussed below are assigned a default value. If a named *file* does exist, **admin** changes the file's parameters that correspond to specified options; other parameters are left as is.

If you have named a directory in the command line, **admin** behaves as though each file in the directory was a named *file*, except that it silently ignores non-SCCS and unreadable files. If you have given a dash (–) in place of the filename, **admin** reads the standard input, taking each line to be the name of an SCCS file to be processed. Again, it silently ignores non-SCCS and unreadable files.

OPTIONS

Each of the options below assumes that only one named file is to be processed, since the effects of the arguments apply independently to each named file. On options beginning with **–f** and followed by an argument, the specified flag (and value, if applicable) is placed in the SCCS file. You may supply several of these **–f** options on a single **admin** command line.

- n** Create a new SCCS file from the contents of the file specified.
- i[*name*]** Specify *name* as the name of the file from which the text for a new SCCS file is to be taken. The text comprises the first delta of the specified file. If you omit *name*, **admin** obtains the text for the SCCS file by reading the standard input until an end-of-file is encountered. This option allows you to create only one SCCS file at a time. Without it, the SCCS file is created empty.
- rr*el*** Insert the initial delta into the specified release (the default is release 1). You may use this option only if the **–i** option is also present. Note that the default level of initial deltas is always 1; thus, the naming of initial deltas defaults to 1.1.
- t[*name*]** Use the name of the file specified for descriptive text for the SCCS file. If you use this option with **–n** and/or **–i**, you must also supply the descriptive text filename. For existing SCCS files, using this option

- without specifying a filename removes descriptive text (if any) from the SCCS file. If a filename is specified, any descriptive text in the SCCS file is replaced by existing text in the named file.
- fb** Allow use of the **-b** keyletter on a **get(1)** command to create branch deltas.
- fcceil** Specify the highest release ("ceiling") that may be retrieved by a **get(1)** command for editing. The number should be less than or equal to 9999. The default is 9999.
- fffloor** Specify the lowest release ("floor") that may be retrieved by a **get(1)** command for editing. The number should be greater than zero but less than 9999. The default is 1.
- fdSID** Indicate the default delta number (SID) to be used by a **get(1)** command.
- fi[str]** Treat the "No id keywords (ge6)" message issued by **get(1)** or **delta(1)** as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must be identical to the given string; however, the string must contain a keyword, and no embedded newlines.
- fj** Allow concurrent **get(1)** commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
- flist** Specify a *list* of releases to which deltas can no longer be made (**get -e** against one of these "locked" releases fails). The *list* has the following syntax:
- `<list> ::= <range> | <list> , <range>`
`<range> ::= RELEASE NUMBER | a`
- The character *a* in the *list* is equivalent to specifying *all releases* for the named SCCS file.
- fn** Force **delta(1)** to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

- fqtext** Substitute user-definable text for all occurrences of the %Q% keyword in SCCS file text retrieved by `get(1)`.
- fmmod** Substitute the *module* name of the SCCS file for all occurrences of the %M% keyword in SCCS file text retrieved by `get(1)`. If the **-m** option is not specified, the value assigned is the name of the SCCS file with the leading *s.* removed.
- ftype** Substitute the *type* of module in the SCCS file for all occurrences of %Y% keyword in SCCS file text retrieved by `get(1)`.
- fv[pgm]** Force `delta(1)` to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity-checking program. (If this flag is set when creating an SCCS file, the **-m** option must also be used, even if its value is null.)
- dflag** Remove (delete) the specified *flag* from an SCCS file. The **-d** option may be specified only when processing existing SCCS files. Several **-d** options may be supplied on a single **admin** command. See the options beginning with a **-f** (e.g., **-fb**, **-fj**, etc) for allowable *flag* names.
- l_{list}** Specify a *list* of releases to be “unlocked.” See the **-fl_{list}** option for a description of the *l* flag and the syntax of a *list*.
- a_{login}** Indicate a *login* name, or numerical UNIX system group ID, to be added to the list of users that may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several of these options may be used on a single **admin** command line. As many *logins*, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. Persons with *login* or group ID preceded by an exclamation point (!) are denied permission to make deltas.
- e_{login}** Indicate a *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **-e** options may be used on a single **admin** command line.
- y[comment]** Insert the *comment* text into the SCCS file as a comment for the initial delta in a manner identical to that of `delta(1)`. Omitting the **-y** option results in a default comment line being inserted in the form:

date and time created YY/MM/DD HH:MM:SS by *login*

The **-y** option is valid only if the **-i** and/or **-n** options are specified (i.e.,

a new SCCS file is being created).

- m [mrlist]** Insert the list of Modification Request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to **delta(1)**. The **v** flag must be set and the MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). Diagnostics occur if the **v** flag is not set or MR validation fails. See the **-fv** option described above.
- h** Check the structure of the SCCS file and compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. This option produces appropriate error diagnostics. It also inhibits writing on the file, so that it nullifies the effect of any other options supplied. Therefore, it is only meaningful when processing existing files.
- z** Recompute the SCCS file check-sum and store it in the first line of the SCCS file (refer to the **-h** option above). Note that using this option on a truly corrupted file may prevent future detection of the corruption.

EXAMPLE

To create a new SCCS file (called *sccsfile1*) from *file1*, type the following:

```
# admin -ifile1 s.sccsfile1
```

CAUTIONS

Directories containing SCCS files should be 755 mode, so that only the owner can modify SCCS contained in the directories. The mode of SCCS files themselves should be mode 444, to prevent modifications except by SCCS commands.

If you have to patch an SCCS file, the owner must change the file's mode to 644, to allow use of **ed(1)**. The edited file should always be processed by an **admin -h** to check for corruption, followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is then recommended to ensure that the SCCS file is valid.

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

delta(1), **ed(1)**, **get(1)**, **help(1)**, **prs(1)**, **what(1)**, **sccsfile(4)**.

DOMAIN/IX Support Tools Guide (for general information on SCCS).

NAME

ar – archive and library maintainer for portable archives

USAGE

ar *key* [*posname*] *afile* [*name*] ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Although it is mainly used to create and update library files needed by the link editor, **ar** can be used for other similar purposes. The file headers used by **ar** are printable ASCII characters. If an archive comprises printable files, the entire archive is printable.

When **ar** creates an archive, it produces headers in a format portable across all machines. The portable archive format and structure is described in detail in **ar(4)**. The link editor uses the archive symbol table to effect multiple passes over libraries of object files in an efficient manner. The link editor is further described under **ld(1)**.

Ar creates and maintains an archive symbol table only when there is at least one object file in the archive. The archive symbol table is in a specially named file that is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever **ar** creates or updates the contents of such an archive, it also rebuilds the symbol table.

Key is an optional dash (-), followed by one character from the **drqtpmx** set, optionally concatenated with one or more characters from the **vuaibcls** set. *Posname* is the name of an optional positioning character. *Afile* is the archive file.

KEY CHARACTERS

- | | |
|----------|---|
| c | Suppress the message produced by default when <i>afile</i> is created. |
| d | Delete named files from the archive file. |
| l | Place temporary files in the local current working directory, not in the directory specified by the environment variable TMPDIR or in the default directory called <i>/tmp</i> . |
| m | Move named files to the end of the archive. If a positioning character is present, the <i>posname</i> argument must be present to specify where the files should be moved. |
| p | Print named files in the archive. |
| q | Quickly append named files to the end of the archive file, without checking to see whether added members already exist in the archive. Optional positioning characters are invalid. This option is useful only during piece-by-piece creation of a large archive. |
| r | Replace named files in the archive file. If the optional character u is |

also used, replace only those files with modification dates later than the archive files. If an optional positioning character from the set *abi* is used, the *posname* argument must be present to specify that new files are to be placed after (a) or before (b or i) *posname*. Otherwise, new files are placed at the end.

- s Force the regeneration of the archive symbol table, even if *ar* is not invoked with a command that modifies the archive contents. This option is useful after the *strip(1)* command has been executed on the archive.
- t Print a table of contents of the archive file. If no names are given, table all files in the archive. If names are given, table only those files.
- v Provide a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with *t*, give a long listing of all information about the files. When used with *x*, precede each file with a name.
- x Extract the named files. If no names are given, extract all files in the archive. In neither case does *x* alter the archive file.
- u Update archive element if file is newer.
- o Restore original file times when extracting from the archive (this is ignored when used with Apollo load libraries).

CAUTIONS

If the same file is mentioned twice in an argument list, it may be put into the archive twice.

FILES

*/tmp/ar** *temporaries*

RELATED INFORMATION

arcv(1), *ld(1)*, *lorder(1)*, *strip(1)*, *tmpnam(3S)*, *ar(4)*.

NAME

arcv – convert archive files from PDP-11 to common archive format

USAGE

arcv *infile outfile*

DESCRIPTION

Arcv converts source archive files from PDP-11 format to UNIX System V portable archive format. It converts the input archive file *infile* (but not the individual members of *infile*) to an equivalent output archive file *outfile*.

FILES

*/tmp/arcv**

RELATED INFORMATION

ar(1), **ar**(4).

NAME

asa – interpret ASA carriage control characters

USAGE

asa [*files*]

DESCRIPTION

Asa interprets the output of FORTRAN programs that use ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no filenames are supplied. The first character of each line is assumed to be one of the following control characters:

' '	(Blank) Single newline before printing;
0	Double newline before printing;
1	New page before printing;
+	Overprint previous line.

If a line begins with anything other than the above characters, asa automatically interprets it as beginning with a ' ', and produces an appropriate diagnostic on the standard error. It never prints the first character of a line, and it always forces the first line of each input file to start on a new page.

RELATED INFORMATION

ratfor(1).

NAME

at, **batch** – execute commands at a later time

USAGE

at *time* [*date*] [+ *increment*]

at -r*job*...

at -l [*job* ...]

batch

DESCRIPTION

At and **batch** read commands from standard input to be executed at a later time. **At** allows you to specify when the commands should be executed, while jobs queued with **batch** executes when the system load level permits.

Standard output and standard error output are mailed to you, unless you redirect them elsewhere. The Shell environment variables, current directory, and file-creation mode mask setting are retained when you execute either **at** or **batch**. Open file descriptors, traps, and priority are lost.

You can use **at** if your name appears in the file */usr/lib/cron/at.allow*. If that file does not exist, the file */usr/lib/cron/at.deny* determines whether you are allowed to use **at**. If */usr/lib/cron/at.allow* does not exist and */usr/lib/cron/at.deny* is empty, global use is allowed. If neither file exists, only *root* can submit a job. The allow/deny files consist of one user name per line.

Batch submits a batch job. It is almost equivalent to the command **at now**, but **batch** goes into a different queue and responds much earlier with any error messages.

OPTIONS

The following options apply to **at** *only*:

[*time*] [+ *increment*]

Specify *time* when commands are to be executed. One- and two-digit numbers indicate hours, four-digit numbers show hours and minutes. You may alternately specify the time as two numbers separated by a colon, meaning *hour:minute*. You may also append an *am* or *pm* suffix; otherwise, the commands will assume a 24-hour clock. The suffix *zulu* may be used to indicate GMT. The special names of *noon*, *midnight*, *now*, and *next* are also recognized.

You may specify an optional *date* as either a month name followed by a day number (and possibly a year number preceded by an optional comma), or a day of the week (fully spelled or abbreviated to three characters). Two special “days,” *today* and *tomorrow* are recognized. If you have not provided a *date*, *today* is assumed if the given hour is greater than the current hour, and

tomorrow if it is less. If the given month is less than the current month (and no year is given), *next year* is assumed.

The optional *increment* is simply a number suffixed by one of the following: *minutes*, *hours*, *days*, *weeks*, *months*, or *years*. (The singular form is also accepted.)

-rjob Remove jobs previously scheduled by **at** or **batch**. You can only remove your own jobs unless you are the super-user.

-l[job] Report all jobs (by job number) scheduled for the invoking user.

EXAMPLES

The **at** and **batch** commands read from standard input the commands to be executed at a later time. **Sh(1)** provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
nroff filename >outfile
<CONTROL-Z>
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a Shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
nroff filename 2>&1 >outfile | mail loginid
!
```

To have a job reschedule itself, invoke **at** from within the Shell procedure, by including code similar to the following within the Shell file:

```
echo sh Shellfile | at 1900 thursday
```

Some examples of simple, yet valid, **at** command lines are shown here:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

FILES

<i>/usr/lib/cron</i>	main cron directory (link to ' <i>node_data/cron</i>)
<i>/usr/spool/cron</i>	cron spool area (link to ' <i>node_data/cron</i>)
<i>/usr/spool/cron/atjobs</i>	Spool area for atjob files.

*/usr/lib/cron/at.** Permissions files (*at.allow*, *at.deny*)

/usr/lib/cron/log Log file.

DIAGNOSTICS

Complains about various syntax errors and times out of range.

RELATED INFORMATION

kill(1), mail(1), nice(1), ps(1), sh(1), cron(1M),

NAME

awk – pattern scanning and processing language

USAGE

awk [**-F** *c*] [**-F** *script* | *prog*] [*parameters*] [*files*]

DESCRIPTION

Awk scans input *files* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog*, it performs an associated action when a line matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as **-f scriptfile**. Enclose the *prog* string in single quotes (' ') to protect it from the Shell.

Parameters, in the form *x=... y=...* etc., may be passed to **awk**.

Awk reads files in the order in which they appear on the command line. If you do not specify any files or if you use a dash (–) in place of any filenames, it reads the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is composed of fields separated by white space. This default can be changed by using the **FS** variable name (see below). The fields are denoted *\$1*, *\$2*, ...; *\$0* refers to the entire line.

A pattern-action statement has the following form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement is one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next # skip remaining patterns on this input line
exit # skip the rest of the input
```

Statements are terminated by semicolons, newlines, or right braces. An empty *expression-list* stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, –, *, /, %, and concatenation (indicated by a blank). The following C operators are also valid in expressions: ++, —, +=, -=, *=, /=, and %= . Variables may be scalars, array elements (denoted by *x[i]*),

or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are placed in double quotes (" ").

The *print* statement prints its arguments on the standard output (or on a file if *>expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the *printf(3S)* format.

The built-in function called *length* returns the length of its argument taken as a string, or of the whole line if no argument exists. There are also built-in functions known as *exp*, *log*, *sqrt*, and *int*. The last function truncates its argument to an integer; *substr(s, 'm, 'n)* returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf(fmt, 'expr, 'expr, '...)* formats the expressions according to the *printf(3S)* format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (*!*, *||*, *&&*, and parentheses) of regular and relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

Regular expressions must be surrounded by slashes and appear as in *egrep*, which appears under *grep(1)*. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

where a *relop* is any of the six relational operators in C, and a *matchop* is either a tilde (*~*) for *contains*, or an exclamation point and a tilde (*!~*) for *does not contain*. A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns *BEGIN* and *END* may be used to capture control before the first input line is read and after the last. *BEGIN* must be the first pattern; *END* must be the last.

A single character *c* may be used to separate the fields by starting the program with:

BEGIN { FS = c }

or by using the *-Fc* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default newline); and OFMT, the output format for numbers (default `%.6g`).

EXAMPLES

To print lines longer than 72 characters, specify the following:

```
length > 72
```

To print the first two fields in opposite order, use this:

```
{ print $2, $1 }
```

To add the first column, and then print the sum and average, specify this:

```
{ s += $1 }  
END { print "sum is", s, " average is", s/NR }
```

To print fields in reverse order, specify the following:

```
{ for (i = NF; i > 0; --i) print $i }
```

To print all lines between start/stop pairs, use this:

```
/start/, /stop/
```

To print all lines whose first field is different from the previous one, use this:

```
$1 != prev { print; prev = $1 }
```

To print the file, filling in page numbers starting at 5, specify the following:

```
/Page/ { $2 = n++; }  
      { print }
```

command line: `awk -f program n=5 input`

CAUTIONS

Input white space is not preserved on output if fields are involved.

No explicit conversions exist between numbers and strings. To force an expression to be treated as a number, add zero to it. To force it to be treated as a string, concatenate the null string (" ") to it.

RELATED INFORMATION

`grep(1)`, `lex(1)`, `sed(1)`, `malloc(3C)`, `printf(3S)`.

NAME

banner – make posters

USAGE

banner *string*

DESCRIPTION

Banner prints the specified *string* (up to 15 characters long) in large letters on the standard output.

RELATED INFORMATION

echo(1).

NAME

basename, **dirname** – deliver portions of pathnames

USAGE

basename *string* [*suffix*]
dirname *string*

DESCRIPTION

Basename deletes from *string* any prefix ending with a slash (/), as well as any *suffix* present. It then prints the result on the standard output. **Basename** is normally used inside substitution marks (``) within shell procedures.

Dirname delivers all but the last level of the pathname in *string*.

EXAMPLES

To remove a specified suffix of *c* from the *prog.c* string, and produce *prog* as the resulting output, type the following:

```
basename prog.c .c
```

To set the Shell variable *NAME* to */usr/cmd*, specify this:

```
NAME= `dirname /usr/cmd/cat.c`
```

CAUTIONS

The **basename** of / is null and considered an error.

RELATED INFORMATION

sh(1).

NAME

bc – arbitrary-precision arithmetic language

USAGE

bc [-c] [-l *name*] [*file* ...]

DESCRIPTION

Bc is an interactive processor for a language that resembles C, but provides unlimited precision arithmetic. It takes input from any *file* given, then reads the standard input.

Bc is actually a preprocessor for **dc**(1), which it usually invokes automatically.

PROGRAM SYNTAX

The syntax for bc programs is shown below (L = letters a–z, E = expression, S = statement).

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. As in **dc**(1), an assignment to *scale* influences the number of digits to be retained on arithmetic operations. An assignment to *ibase* or *obase* sets the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. “Auto” variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

Comments

Enclosed in /* and */.

Names

Simple variables: L

Array elements: L [E]

The words *ibase*, *obase*, and *scale*

Other Operands

Arbitrarily long numbers with optional signs and decimal points

(E)

sqrt (E)

length (E) Number of significant decimal digits

scale (E) Number of digits right of decimal point

L (E , ... , E)

Operators

+ - * / % ^ (% is remainder; ^ is power)

++ — (Prefix and postfix; apply to names)

== <= >= != < >

= =+ =- =* =/ =% =^

Statements

```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function Definitions

```
define L ( L ,..., L ) {
    auto L, ... , L
    S; ... S
    return ( E )
}
```

Functions in Math Library

```
s(x)    sine
c(x)    cosine
e(x)    exponential
l(x)    log
a(x)    arctangent
j(n,x)  Bessel function
```

All function arguments are passed by value.

OPTIONS

-c Compile only. Send the input of dc(1) to standard output.

-l *name* Specify *name* as the name of an arbitrary precision math library to be used.

EXAMPLE

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; 1==1; i++){
        a = a*x
        b = b*i
        c = a/b
```

```
        if(c == 0) return(s)
        s = s+c
    }
}
```

defines a function to compute an approximate value of the exponential function and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function for the first ten integers.

CAUTIONS

The double ampersand (&&) and double pipe (||) characters are not yet valid operators for use with **bc**.

A *for* statement must have all three E's.

Quit is interpreted when read, but not when executed.

FILES

<i>/usr/lib/lib.b</i>	mathematical library
<i>/usr/bin/dc</i>	desk calculator proper

RELATED INFORMATION

dc(1).

NAME

bdiff – big diff

USAGE

bdiff *file1 file2* [*n*] [-s]

DESCRIPTION

Bdiff compares two files and shows which lines differ. It produces results similar to **diff(1)**, but allows processing of much larger files.

Bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes **diff** upon corresponding segments. The default value of *n* is 3500, but you may optionally supply an alternate numeric value. This is useful when 3500-line segments are too large for **diff**, causing it to fail. If you supply a dash (-) in place of *file1* (*file2*), **bdiff** reads the standard input.

The output of **bdiff** is exactly that of **diff**, with line numbers adjusted to account for the segmenting of the files (i.e., appearing as if the files had been processed whole). Note that because of the segmenting of the files, **bdiff** does not necessarily find the smallest sufficient set of file differences.

OPTIONS

-s Silent. Prohibits printing of diagnostics by **bdiff** (but does not suppress possible exclamations by **diff**).

FILES

/bin/**bdiff**
/tmp/bd????

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

diff(1).

NAME

bfs – big file scanner

USAGE

bfs [-] *name*

DESCRIPTION

Bfs is similar to ed(1), except that it is read-only and processes much larger files. Maximum file size is 1024K bytes. Files can be up to 32K lines in length, with as many as 512 characters per line, including newline (255 characters for 16-bit machines). Bfs is usually more efficient than ed(1) for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where csplit(1) can be used to divide it into more manageable pieces for editing.

Unless you specify a dash (-) option, bfs prints the size of the file being scanned. If P and a carriage return are typed, it prompts input with an asterisk (*). Prompting can be turned off again by inputting another P and carriage return. Note that messages are given in response to errors if prompting is turned on.

EXPRESSIONS

Bfs supports all address expressions described under ed(1). Thus, regular expressions may be surrounded with a slash (/) and question mark (?). In addition, a greater-than character (>) indicates a downward search without wrap-around, and a less-than character (<) indicates an upward search without wrap-around. There is a slight difference in mark names: only the letters a through z may be used, and all 26 marks are remembered.

COMMANDS

The e, g, k, p, q, v, w, =, ! and null commands operate as described under ed(1). Commands such as ---, +++-, +++=, -12, and +4p are accepted. Note that 1,10p and 1,10 will both print the first ten lines. The f command only prints the name of the file being scanned; there is no *remembered* filename. The w command is independent of output diversion, truncation, or crunching (see the xo, xt, and xc commands below). The following commands are also available:

xf *file*

Take further commands from the named *file*. When an end-of-file is reached, an interrupt signal is received, or an error occurs, reading resumes with the file containing the xf. The xf commands may be nested to a depth of ten.

xn

List the marks currently in use (marks are set by the k command).

xo [*file*]

Divert further output from the p and null commands to the named *file*, which, if necessary, is created in mode 666. If *file* is missing, divert output to the standard

output. Each diversion truncates or creates a file.

: *label*

Position a *label* in a command file. Ignore blanks between the colon (:) and the start of the *label*. Terminate *label* with newline. May also be used to insert comments into a command file, since labels need not be referenced.

(. . .)xb/*regular expression*/*label*

Make a jump (either upward or downward) to *label* if successful. Will fail under any of the following conditions:

1. Neither address is between 1 and \$.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

If successful, set the current line, indicated by a period (.), to the line matched. Then, jump to *label*. This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that **xb/^/*label*** is an unconditional jump.

The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe (|), only a downward jump is possible.

xt *number*

Truncate output from the **p** and null commands to a maximum of *number* characters (initially 255).

xv[*digit*][*spaces*][*value*]

Specify *digit* as the variable name following **xv**. The commands **xv5100** or **xv5 100** both assign the value 100 to the variable 5. The command **Xv61,100p** assigns the value 1,100p to the variable 6. To reference a variable, put a percent sign (%) in front of the variable name. For example, using the above assignments for variables 5 and 6:

```
1,%5p
1,%5
%6
```

all print the first 100 lines.

To globally search for the characters 100 and print each line containing a match, type:

```
g/%5/p
```

To escape the special meaning of a percent sign (%), a backslash (\) must precede it.

The following can be used to match and list lines containing *printf* of characters, decimal integers, or strings:

```
g/".*\%[cds]/p
```

Another feature of the *xv* command is that the first line of output from a UNIX system command can be stored into a variable. The only requirement is that the first character of *value* be an exclamation point (!). For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

puts the current line into variable 5, prints it, and increments the variable 6 by one. To escape the special meaning of an exclamation point (!) as the first character of *value*, precede it with a slash (\).

```
xv7!\date
```

stores the value *!date* into variable 7.

xbz label

xbn label

Test the last saved return code from the execution of a UNIX system command (*!command*) or non-zero value, respectively, to the specified label. For example, to search for the next five lines containing the string called "size", you might do the following:

```
xv55
: 1
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn 1
xv45
: 1
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

xc [switch]

If *switch* is 1, output from the *p* and null commands is crunched; if *switch* is 0 it is not. Without an argument, *xc* reverses *switch*. Initially *switch* is set for no

crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

RELATED INFORMATION

csplit(1), ed(1), regcmp(3X).

NAME

bs – a compiler/interpreter for modest-sized programs

USAGE

bs [*file* [*args*]]

DESCRIPTION

Bs is a remote descendant of BASIC and SNOBOL4, with some C language added. It is used for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the **trace** and **dump** statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see **compile** and **execute** below). Unless the final operation is an assignment, the result of an immediate expression statement is printed.

Bs programs are composed of input lines. If the last character on a line is a backslash (\), the line is continued. **Bs** accepts lines of the following form:

statement
label statement

A **bs statement** is either an expression, or a keyword followed by zero or more expressions. Some keywords (**clear**, **compile**, **!**, **execute**, **include**, **ibase**, **obase**, and **run**) are always executed as they are compiled. An *expression* is executed for its side effects (value, assignment, or function call). The details of expressions follow the description of statement types below.

A *label* is a *name* (see **EXPRESSION SYNTAX** below) followed by a colon. A *label* and a *variable* can have the same name.

STATEMENT TYPES

- | | |
|--------------------------------------|--|
| break | Exit from the innermost <i>for/while</i> loop. |
| clear | Clear the symbol table and compiled statements. This statement type is executed immediately. |
| compile [<i>expression</i>] | Compile succeeding statements and override the immediate execution default. Evaluate the optional <i>expression</i> and use it as a filename for further input. Associate a clear with this latter case. This statement type is executed immediately. |

- continue** Transfer to the loop-continuation of the current **for/while** loop.
- dump** [*name*] Print the name and current value of every nonlocal variable. Optionally, if a specific *name* is used, report only on that named variable. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.
- exit** [*expression*]
 Go back to system level, returning the *expression* as process status.
- execute** Change to immediate execution mode (an interrupt has a similar effect). This does not cause stored statements to execute (see **run** below).
- for** *name* = *expression* *expression* *statement*
for *name* = *expression* *expression*
 ...
next
- for** *expression* , *expression* , *expression* *statement*
for *expression* , *expression* , *expression*
 ...
next Repetitively execute a *statement* (first form) or a group of *statements* (second form) under control of the variable specified by *name*. The variable takes on the value of the first *expression*, then is incremented by one on each loop, not to exceed the value of the second *expression*. The third and fourth forms require three *expressions* separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).
- fun** *f* [*a*, ...] [*v*, ...]
 ...
- nuf** Define the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.
- freturn** Signal the failure of a user-written function. See the interrogation operator (?) below. If interrogation is not present, **freturn** merely returns zero. When interrogation *is* active, **freturn** transfers to that expression (possibly bypassing intermediate function returns).
- goto** *name* Pass control to the internally-stored statement with the matching label.
- ibase** *N* Set the input base (radix) to *N*. The only supported values for *N* are 8, 10 (the default), and 16. Hexadecimal values 10–15 are entered as a-f. A leading digit is required (i.e., f0a must be entered as 0f0a). This

statement type is executed immediately.

if expression statement

if expression

...

[else

...]

fi

Execute *statement* (first form) or group of *statements* (second form) if the expression evaluates to non-zero. The 0 and "" (null) strings evaluate as zero. In the second form, an optional else allows for a group of *statements* to be executed when the first group is not. The only *statement* permitted on the same line with an else is an if. Only other fi *statements* can be on the same line with an fi. The elision of else and if into an elif is supported. Only a single fi is required to close an if ... elif ... [else ...] sequence.

include expression

Expression must evaluate to a filename. The file must contain bs source statements. Such statements become part of the program being compiled. Include statements may not be nested.

obase N

Set the output base to *N* (see *ibase* above).

onintr label

onintr

Provide program control of interrupts. In the first form, control passes to the label given, just as if a goto had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt causes bs to terminate.

return [expression]

Evaluate the *expression* and pass the result back as the value of a function call. If no expression is given, zero is returned.

run

Reset the random number generator and pass control to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop

Stop execution of internal statements. Bs reverts to immediate mode.

trace [expression]

Control function tracing. If *expression* is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace expression* value.

while expression statement

while

expression

- ...
next While is similar to **for**, except that only the conditional *expression* for loop-continuation is given.
- ! Shell command**
 Make an immediate escape to the shell.
- #...** Interject commentary in a program.

EXPRESSION SYNTAX

- name** Specify a variable. Composed of an upper- or lowercase letter, optionally followed by letters and digits. Only the first six characters of a *name* are significant. Except for those declared in **fun** statements, all names are global to the program. A name can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function **open()** below).
- name ([expression [,expression] ...])**
 Specify the *name* of a function. This *name* should be followed by the arguments in parentheses, separated by commas. Except for built-in functions (listed below), the name must be defined with a **fun** statement. Arguments to functions are passed by value.
- name [expression [,expression] ...]**
 Reference either arrays or tables (see built-in **table** functions below). For arrays, each *expression* is truncated to an integer and used as a specifier for the *name*. The resulting array reference is syntactically identical to a name; thus, *a*[1,2] is the same as *a*[1][2]. The truncated expressions are restricted to values between 0 and 32767.
- number**
 Represent a constant value. Written in FORTRAN style, this syntax contains digits and an optional decimal point. It may also contain a scale factor consisting of an *e* followed by perhaps a signed exponent.
- string** Delimit character *strings* by characters in quotes. The backslash (\) escape character allows the double quote (\'), newline (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. Otherwise, a backslash stands for itself.
- (expression)**
 Use parentheses to alter the normal order of evaluation.
- (expression ,expression [,expression ...]) [expression]**
 Use the bracketed *expression* as a subscript to select a comma-separated *expression* from the parenthesized list. Number list elements from the left, starting at zero. The following expression has the value of *True* if the comparison is true:

(False, True)[a == b]

? *expression*

Test for the success of the *expression*, rather than its value. This is useful for testing end-of-file, the result of the `eval` built-in function. Also helpful for checking the return from user-written functions (see `freturn`). An interrogation “trap” (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

-*expression*

Negate the expression.

++*name* Increment the value of the variable (or array reference). The result is the new value.

--*name* Decrement the value of the variable. The result is the new value.

!*expression* Logically negate the expression. Make sure to use the Shell escape.

expression operator expression

Abbreviate the common functions of two arguments by separating the two arguments by an operator denoting the function. Except for the assignment, concatenation, and relational operators, convert both operands to numeric form before the function is applied.

BINARY OPERATORS

The binary operators described below are listed by increasing precedence.

= Assignment operator. The left operand must be a name or an array element. Result is the right operand. Assignment binds right to left, all other operators bind left to right.

_ (Underscore); concatenation operator.

& | Logical *and* (&) has result zero if either of its arguments are zero. This operator has result one if both of its arguments are non-zero. Logical *or* (|) has result zero if both of its arguments are zero, and result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == !=

Relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to). Return one if their arguments are in the specified relation; otherwise, return zero. Those operators at the same level extend as follows: $a > b > c$ is the same as $a > b$ & $b > c$. A string comparison is made if both operands are strings.

+ -

Add and subtract.

* / %

Multiply, divide, and remainder.

^

Exponentiate.

BUILT-IN FUNCTIONS FOR DEALING WITH ARGUMENTS

arg(*i*) Value of the *i*th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns *bs*).

narg() Return the number of arguments passed. At level zero, it will return the command argument count.

BUILT-IN MATHEMATICAL FUNCTIONS

abs(*x*) Absolute value of *x*.

atan(*x*) Arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.

ceil(*x*) Return the smallest integer not less than *x*.

cos(*x*) Cosine of *x* (radians).

exp(*x*) Exponential function of *x*.

floor(*x*) Return the largest integer not greater than *x*.

log(*x*) Natural logarithm of *x*.

rand() Uniformly distributed random number between zero and one.

sin(*x*) Sine of *x* (radians).

sqrt(*x*) Square root of *x*.

BUILT-IN FUNCTIONS FOR STRING OPERATIONS

size(*s*) Return the size (length in bytes) of *s*.

format(*f*,*a*) Return the formatted value of *a*. Assume *f* to be a format specification in the style of **printf(3S)**. Only the %...f, %...e, and %...s types are safe.

index(*x*,*y*) Return the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(*s*,*f*,*t*) Translate characters of source *s* from matching characters in *f* to a character in the same position in *t*. Copy source characters that do not appear in *f* to the result, unless string *f* is longer than *t*, in which case the source

characters that match in the excess portion of *f* will not appear in the result.

substr(*s*,*start*,*width*)

Return the substring of *s* defined by the *starting* position and *width*.

match(*string*,*pattern*)
mstring(*n*)

Pattern is similar to the regular expression syntax of the *ed*(1) command. The characters *.*, *[*, *]* are special. The caret placed inside brackets *[^]*, the asterisk (***), and the dollar sign (*\$*) are special. The *mstring* function returns the *n*-th ($1 \leq n \leq 10$) substring of the subject that occurred between pairs of the pattern symbols *\(* and *\)* for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with *^*). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\[a-z\]\)") == 6
mstring(1) == "b"
```

BUILT-IN FILE HANDLING FUNCTIONS

open(*name*, *file*, *function*)

close(*name*)

The *name* argument must be a bs variable name (passed as a string). For the *open*, the *file* argument may be one of the following: 1) a 0 (zero), 1, or 2 representing standard input, output or error output, respectively; 2) a string representing a filename; or 3) a string beginning with an exclamation point (!) representing a command to be executed (via *sh -c*). The *function* argument must be either *r* (read), *w* (write), *W* (write without new-line), or *a* (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

access(*s*, *m*)

Execute *access*(2).

ftype(*s*)

Return a single-character file type indication: *f* for regular file, *p* for FIFO (i.e., named pipe), *d* for directory, *b* for block special, or *c* for character special.

TABLES

table(*name*, *size*)

A table in bs is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a bs variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. Bs prints an error message and stops on table overflow.

item(*name*, *i*)

key() The **item** function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the **item** function accesses values, the **key** function accesses the "subscript" of the previous **item** call. The *name* argument should not be quoted. Since exact table sizes are not defined, use the interrogation operator to detect end-of-table. For example:

```
table("t", 100)
```

```
...
```

```
# If word contains "party", the following expression adds one
# to the count of that word:
```

```
++t[word]
```

```
...
```

```
# To print out the the key/value pairs:
```

```
for i = 0, ?(s = item(t, i)), ++i if key() put = key()_"":_s
```

iskey(*name*, *word*)

Test whether the key *word* exists in the table *name* and return one for true or zero for false.

MISCELLANEOUS BUILT-IN FUNCTIONS

eval(*s*)

Evaluate the string argument *s* as a bs expression. This function is handy for converting numeric strings to numeric internal form. It can also be used as a crude form of indirection. For example, the following increments the variable *xyz*:

```
name = "xyz"
eval("++" _ name)
```

In addition, **eval** preceded by the interrogation operator allows you to control bs error conditions. For example, the following returns the value zero if there is no file named "XXX" (instead of halting your program):

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

The following executes a **goto** to the label *L* (if it exists):

```
label="L"  
if !(?eval("goto " _ label)) puterr = "no label"
```

last() In immediate mode, last returns the most recently computed value.

EXAMPLES

This shows how to use bs as a calculator:

```
# bs  
# Distance (inches) light travels in a nanosecond.  
186000 * 5280 * 12 / 1e9  
311.78496  
  
# Compound interest (6% for 5 years on $1,000).  
int = .06 / 4  
bal = 1000  
for i = 1 5*4 bal = bal + bal*int  
bal - 1000  
346.855007  
  
exit  
#
```

The outline of a typical bs program is as follows:

```
# initialize things:  
var1 = 1  
open("read", "infile", "r")  
...  
# compute:  
while ?(str = read)  
...  
next  
# clean up:  
close("read")  
...  
# last statement executed (exit or stop):  
exit  
# last input line:  
run
```


Bs uses the Standard Input/Output package. Here are some input/output examples:

```
# Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

# Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

RELATED INFORMATION

ed(1)
sh(1)
access(2)
printf(3S)

NAME

cal – print calendar

USAGE

cal [[*month*] *year*]

DESCRIPTION

Cal prints a calendar for a specified month and/or year. If neither is specified, cal prints a calendar for the present month only.

Both *year* and *month* must be Arabic numbers. The range for *year* is 1-9999. The range for *month* is 1-12.

EXAMPLES

To print a calendar for the entire year of 1985, type the following:

cal 1985

To print a calendar for December, 1985, type this:

cal 12 1985

CAUTIONS

Specifying cal 85 prints a calendar for the year 85, not for 1985.

The *year* always starts in January.

NAME

calendar – reminder service

USAGE

/usr/bin/calendar [-]

DESCRIPTION

Calendar provides an individual reminder service by consulting the file *calendar* in your log-in directory and printing out lines containing today's or tomorrow's date. You must create the file before *calendar* can successfully run.

A typical line in your *calendar* file may look like this:

12/15 Departmental meeting at 3 p.m.

Calendar recognizes most month-day entries (e.g., 12/15, Dec. 15, december 15), but not day-month items (e.g., 15 December, etc.). On weekends, "tomorrow" extends through Monday.

When an argument is present, *calendar* looks in all users' log-in directories for a file named *calendar* and sends any positive results by *mail*(1).

CAUTIONS

Your *calendar* must be public information for you to get reminder service.

Calendar's idea of "tomorrow" does not account for holidays.

FILES

/usr/lib/calprog to figure out today's and tomorrow's dates

/etc/passwd

*/tmp/cal**

/usr/mail

RELATED INFORMATION

mail(1).

NAME

cat – concatenate and print files

USAGE

cat [**-u**] [**-s**] [**-v** [**-t**] [**-e**]] *file* ...

DESCRIPTION

Cat reads each named *file* in sequence, then writes it on the standard output. If *file* is omitted and **-** is used instead, **cat** reads from standard input.

OPTIONS

- u** Produce unbuffered output.
- s** Ignore files that do not exist.
- v** Make nonprinting characters visible (except for tabs, newlines, and form feeds). Print control characters as **^X** (**CONTROL-X**), the DEL character (octal 0177) as a caret with a question mark (**^?**), and non-ASCII characters (with the high bit set) as **M-x** (where *x* is the character specified by the seven low-order bits).
- t** With the **-v** option, print tabs as **^I**.
- e** With the **-v** option, print a dollar sign (**\$**) at the end of each line (prior to the newline).

Note that **cat** ignores the **-t** and **-e** options if the **-v** option is not specified.

EXAMPLES

To write *file1* on standard output, type the following:

```
# cat file1
```

To write standard input to *file1*, use this command:

```
# cat >file1
```

To write *file1* and *file2* to *file3*, type this:

```
# cat file1 file2 >file3
```

CAUTIONS

Command formats such as **cat file1 file2 >file1** destroy the original data in *file1*. Be careful when using Shell special characters.

RELATED INFORMATION

cp(1), **pr(1)**.

NAME

cb – C program beautifier

USAGE

cb [**-s**] [**-j**] [**-l** *leng*] [*file ...*]

DESCRIPTION

Cb reads C programs from its arguments or from the standard input, and writes them on the standard output with spacing and indentation displaying the structure of the code. Under default options, **cb** preserves all user newlines.

OPTIONS

-s	Canonicalize code to the style of Kernighan and Ritchie in <i>The C Programming Language</i> .
-j	Put split lines back together.
-l <i>leng</i>	Split lines that are longer than <i>leng</i> .

CAUTIONS

Punctuation hidden in preprocessor statements causes indentation errors.

RELATED INFORMATION

cc(1).

NAME

cc – C compiler

USAGE

cc [*options*] ... *file* ...

DESCRIPTION

Cc is the DOMAIN/IX C compiler. It interprets arguments with names ending in *.c* as C source programs, compiles them, and leaves each object program on the file whose name is that of the source with *.o* substituted for *.c*. Cc normally deletes the *.o* file, however, if a single C program is compiled and loaded simultaneously.

The DOMAIN/IX C compiler is really an interface to the DOMAIN C compiler. Not all standard UNIX options are available. Furthermore, some unique options are provided by the DOMAIN C compiler. The various options that can be used with cc are listed below. Any other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program named *a.out*. See *ld(1)* for load-time options. Also refer to the *DOMAIN C Language Reference* for general information.

OPTIONS

- c** Suppress the loading phase of the compilation and force an object file to be produced, even if only one program is compiled.
- w** Suppress warning diagnostics.
- O** Activate the **-OPT** option to the DOMAIN C compiler.
- Bname** Prefix pathname to cc and bind for substitute compiler and binder passes. If name is empty it is taken to be */usr/lib/o*.
- t[p0l]** Find only the preprocessor (*p*), compiler passes (*0*), or binder (*l*) in the files whose names are constructed by a **-B** option. In the absence of a **-B** option, the name is taken to be */usr/lib/n*. The value **-t ""** is equivalent to **-t0l**.
- Wc,arg1,[arg2...]** Hand off the arguments *argi* to pass *c* where *c* is one of [p0l] indicating the pre-processor, compiler or the binder.
- E** Run only the macro preprocessor on the named C programs, and send the result to the standard output.
- P** Run only the macro preprocessor on the named C programs, and leave the result on corresponding files suffixed with *.i*.

- o output** Name the final output file *output*. Leave the *a.out* file undisturbed.
- Dname=def**
-Dname Define the *name* to the preprocessor, as if by *#define*. If no definition is given, define the name as "1".
- Mid** Generate code for a particular class of processor. Values for *id* are:
- | | |
|------|--|
| any | standard M68000 code |
| 160 | |
| 460 | |
| 660 | DSP160, DN460, DN 660 code |
| 90 | |
| 330 | |
| 560 | |
| 570 | |
| 580 | |
| 3000 | DSP90, DN330, DN560, DN570, DN580, DN3000 code |
| PEB | Performance Enhancement Board |
- Uname** Remove any initial definition of *name*.
- Idir** Seek *#include* files with names not beginning with a slash (/). Look first in the directory of the *file* argument; then in directories named in **-I** options; and finally, in directories on a standard list.
- Tsystype** Define the target system type (*systype*) for the compiled object. *Systype* may be one of
- | | |
|--------|---------------------------------------|
| any | <i>version independent</i> |
| bsd4.1 | <i>BSD4.1 UNIX (AUX SR8)</i> |
| bsd4.2 | <i>BSD4.2 UNIX (DOMAIN/IX bsd4.2)</i> |
| sys3 | <i>UNIX System III (AUX SR8)</i> |
| sys5 | <i>UNIX System V (DOMAIN/IX sys5)</i> |
- g** Turn on the **-DBS** option to the DOMAIN C compiler. This produces symbolic debug information.
- p** Not supported.
- f** Not supported.
- S** Not supported.

CAUTIONS

The compiler currently ignores advice to put *char*, *unsigned char*, *short*, or *unsigned short* variables in registers. It previously produced poor, and in some cases incorrect, code for such declarations.

FILES

<i>file.c</i>	input file
<i>file.o</i>	object file
<i>a.out</i>	loaded output
<i>/usr/lib/cc</i>	AEGIS C compiler
<i>/usr/lib/bind</i>	AEGIS binder
<i>/usr/lib/cpp</i>	AEGIS C pre-processor

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

RELATED INFORMATION

ld(1).

COMMENT

The AEGIS C pre-processor (*/usr/lib/cpp*) is invoked only if *p* is given in a *-t* or *-W* switch. The *-W* is a legitimate special case indicating that */usr/lib/cpp* should be run. If *-p* is not given, the C pre-processor in */usr/lib/cc* is run.

NAME

cd – change working directory

USAGE

cd [*directory*]

DESCRIPTION

If *directory* is not specified, **cd** uses the value of the environment variable **\$HOME** as the new working directory. If *directory* specifies a complete path starting with a slash (/), a period (.), or two consecutive periods (..), *directory* becomes the new working directory. If neither case applies, **cd** tries to find the designated directory relative to one of the paths specified by the **\$CDPATH** Shell variable. **\$CDPATH** has the same syntax as, and similar semantics to, the **\$PATH** Shell variable.

Because a new process is created to execute each command, **cd** would be ineffective if written as a normal command; therefore, it is recognized by and is internal to the shell.

EXAMPLES

To change your working directory to directory called *mydata*, type the following:

```
# cd mydata
```

To advance your working directory one level up in the naming hierarchy, use this command:

```
# cd ..
```

RELATED INFORMATION

pwd(1), **sh(1)**, **chdir(2)**.

NAME

cdc – change the delta commentary of an SCCS delta

USAGE

cdc **-rSID** [**-m** [*mrlist*]] [**-y** [*comment*]] *files*

DESCRIPTION

Cdc changes the delta commentary for a specific SID of each named *file*. Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the **-m** and **-y** arguments to the **delta(1)** command.

If a *directory* is named, **cdc** behaves as though each file in the directory is specified as a named file, except that it silently ignores non-SCCS and unreadable files. If a name of **-** is given, **cdc** reads the standard input. It takes each line of the standard input to be the name of an SCCS file to be processed.

Arguments to **cdc**, which may appear in any order, consist of *options* and filenames. All options described below apply independently to each named file.

OPTIONS

- rSID** Specify the SCCS IDentification string (SID) of a delta for which the delta commentary is to be changed.
- m [mrlist]** Supply a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the **-r** option. The SCCS file must have the **v** flag set. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of **delta(1)**. To delete an MR, precede the MR number with an exclamation point (!). If the MR to be deleted exists in the current list of MRs, it is removed and changed into a “comment” line. A list of all deleted MRs is placed in the comment section of the delta commentary. This list is preceded by a comment line stating that the MRs were deleted.

If **-m** is not used, and the standard input is a terminal, the prompt *MRs?* is issued on the standard output before the standard input is read. If the standard input is not a terminal, no prompt is issued. The *MRs?* prompt always precedes the *comments?* prompt (see the **-y** option).

MRs in a list are separated by blanks and/or tab characters. An unescaped newline character terminates the MR list.

Note that, if the **v** flag has a value, it is taken to be the name of

a program (or Shell procedure) that validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, `cdc` terminates and the delta commentary remains unchanged.

`-y [comment]`

Supersede the existing *comment(s)* for the delta specified by the `-r` option. The previous comments are kept, but preceded by a comment line stating that they were changed. A null *comment* has no effect. If `-y` is not specified, and the standard input is a terminal, the prompt *comments?* is issued on the standard output before the standard input is read. If the standard input is not a terminal, no prompt is issued. An unescaped newline character terminates the *comment* text.

EXAMPLE

To add bl78-12345 and bl79-00001 to the MR list, remove bl77-54321, and add the comment "trouble" to delta 1.6 of s.file:

```
# cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file
```

CAUTIONS

If you supply SCCS filenames to `cdc` via the standard input (`-` on the command line), then you must also use the `-m` and `-y` options.

To modify the delta commentary, you must be either the creator of the delta, or the owner of the SCCS file and directory.

FILES

x-file	see <code>delta(1)</code> .
z-file	see <code>delta(1)</code> .

DIAGNOSTICS

Use `help(1)` for explanations.

RELATED INFORMATION

`admin(1)`, `delta(1)`, `get(1)`, `help(1)`, `prs(1)`, `scsfile(4)`.

NAME

cflow— generate C flow graph

USAGE

cflow [-r] [-ix] [-i_] [-dnum] files

DESCRIPTION

Cflow analyzes a collection of files created by C, yacc(1), and lex(1) in an attempt to build a graph charting the external references. (Note that DOMAIN/IX does not currently support cflow analysis of assembler and object files, although some other systems do.)

Files suffixed in .y, .l, .c, and .i are passed through lex(1) and the C preprocessor (bypassed for .i files) as appropriate. They are then run through the first pass of lint(1). The -I, -D, and -U options of the C preprocessor are also understood. The output of all this nontrivial processing is collected and turned into a graph of external references, which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the -i inclusion option), a colon, and its definition.

For information extracted from C source, the definition consists of an abstract type declaration (e.g., *char **), and, delimited by angle brackets, the name of the source file, and the line number where the definition was found. Definitions extracted from object files indicate the filename and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once the definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only <> is printed.

OPTIONS

- | | |
|--------------|--|
| -r | Reverse the “caller:callee” relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee. |
| -ix | Include external and static data symbols. The default is to include only functions in the flow graph. |
| -i_ | Include names beginning with an underscore. The default is to exclude these functions (and data if -ix is used). |
| -dnum | Use the <i>num</i> decimal integer to indicate the depth at which the flow graph is cut off. By default this is a very large number. You may not set the cut-off depth to a nonpositive integer. |

EXAMPLE

As an example, given the following in *file.c*:

```
int    i;

main()
{
    f();
    g();
    f();
}

f()
{
    i = h();
}
```

the command

```
cflow -ix file.c
```

produces the output

```
1      main: int(), <file.c 4>
2          f: int(), <file.c 11>
3              h: <
4                  i: int, <file.c 1>
5                      g: <
```

When the nesting level becomes too deep, the **-e** option of **pr(1)** can be used to compress the tab expansion to something less than every eight spaces.

CAUTIONS

Files produced by **lex(1)** and **yacc(1)** cause the reordering of line number declarations, which can confuse **cflow**. To get proper results, feed **cflow** the **yacc** or **lex** input.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C preprocessor).

RELATED INFORMATION

cc(1), cpp(1), lex(1), lint(1), pr(1), yacc(1).

NAME

chmod – change mode

USAGE

chmod *mode files*

DESCRIPTION

Chmod allows the named *files* to be changed according to *mode*, which may be absolute or symbolic.

An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the following form:

[*who*] *op permission* [*op permission*]

Who is a combination of the letters **u** for owner's permissions), **g** (group), and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be plus (+) to add *permission* to the file's mode, minus (–) to take away *permission*, or equal (=) to assign *permission* absolutely (reset all other bits).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group IDs), and **t** (save text); **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g**, and **t** only works with **u**.

EXAMPLES

To deny others write permission in *file1*, use the following command:

chmod o–w file1

To make *file1* executable, type this:

```
chmod +x file1
```

To allow the owner of *file1* read, write, and execute permission to *file1* -- and the group and others only read permission -- use this command:

```
# chmod 744 file1
```

CAUTIONS

The DOMAIN system's single-level store requires that all files be mappable and, therefore, readable by the OS. This means that DOMAIN/IX does not recognize execute-only or write-only files. For example, if you type `chmod 111 foo`, DOMAIN/IX automatically sets read permissions for the owner as follows:

```
-r-xr-xr-x 1 owner  unix      5 May 22 11:47 foo
```

Also, if you type `chmod 222 foo`, DOMAIN/IX automatically sets read permissions for owner as follows:

```
-rw-rw-rw- 1 harper sys      5 May 22 11:50 foo
```

Only the owner of a file (or the super-user) may change its mode.

To set the group ID, the group associated with the file must correspond to your current group ID.

RELATED INFORMATION

`ls(1)`, `chmod(2)`.

NAME

chown, chgrp – change owner or group

USAGE

chown *owner file ...*

chgrp *group file ...*

DESCRIPTION

Chown changes ownership of a *file* to a specified individual *owner*. The *owner* argument may be either a decimal user ID or a log-in name found in the password file.

Chgrp changes the group ID for the *file* to a specified *group*. The *group* argument may be either a decimal group ID or a group name found in the group file.

EXAMPLES

To change ownership of *file1* from the current individual owner to ownership by someone with the log-in name of *mary*, use this command:

```
# chown mary file1
```

To change the group identification of *file2* to a group ID of 7, type the following:

```
# chgrp 7 file2
```

CAUTIONS

Unless you are the super-user, invoking **chown** clears the set-user-ID bit (04000) and set-group-ID bit (02000) of the file mode.

The DOMAIN file system requires that files that are protected for write-only or execute-only by **chown(1)** also be made available for read access.

FILES

/etc/passwd

/etc/group

RELATED INFORMATION

chmod(1), **chown(2)**, **group(4)**, **passwd(4)**.

NAME

cmp – compare two files

USAGE

cmp [-l] [-s] *file2*

DESCRIPTION

Cmp makes comparisons between files and indicates the byte and line number where differences occur. If you use a dash (–) in place of *file1*, **cmp** uses the standard input. Under the default options, **cmp** makes no comment if the files are the same. If one file is an initial subsequence of the other, that fact is noted.

OPTIONS

-l	Print the byte number (decimal) and the differing bytes (octal) for each difference.
-s	Print nothing for differing files; return codes only.

DIAGNOSTICS

Returns exit code 0 for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

RELATED INFORMATION

comm(1), **diff(1)**.

NAME

col – filter reverse line-feeds

USAGE

col [**-bfp**x]

DESCRIPTION

Col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7), and by forward and reverse half-line feeds (ESC-9 and ESC-8). **Col** is particularly useful for filtering multicolumn output from **nroff**(1), and output resulting from use of the **tbl**(1) preprocessor.

Although **col** accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. In addition, **col** normally converts white space to tabs on output whenever possible to shorten printing time. Finally, **col** ignores any escape sequences that it does not recognize. Each of these treatments, however, may be suppressed by using the options described below.

Col assumes the ASCII control characters SO (N016) and SI (N017) to start and end text in an alternate character set. It remembers the character set to which each input character belongs, and on output generates SI and SO characters as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, newline, SI, SO, VT (N013), and ESC followed by 7, 8, or 9. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other nonprinting characters are ignored.

The input format accepted by **col** matches the output produced by **nroff**(1) with the **-T37** option. Use **-T37** (and the **-f** option of **col**) if the output of **col** is to be directed to a device that can interpret half-line motions.

OPTIONS

- b** Assume that the output device in use is not capable of backspacing. If two or more characters are to appear in the same place, output only the last one read.
- f** May contain forward half-line feeds (ESC-9) in output, but not any reverse line motion.
- x** Leave white space as is; do not convert to tabs.
- p** Do not ignore unrecognizable escape sequences found in input. Output them as regular characters, subject to overprinting from reverse line motions. Note: Do not use this option unless you are fully aware of

the textual position of the escape sequences.

CAUTIONS

Col cannot back up more than 128 lines.

Local vertical motions that may result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

Col allows a maximum of 800 characters, including backspaces, on a line.

RELATED INFORMATION

nroff(1), tbl(1).

NAME

comb – combine SCCS deltas

USAGE

comb [**-o**] [**-s**] [**pSID**] [**-clist**] *files*

DESCRIPTION

Comb generates a Shell procedure that reconstructs the given SCCS *files*. The reconstructed files are usually smaller than the original *files*. The arguments may be specified in any order, but options apply to all named SCCS files. If a directory is named, **comb** behaves as though each file in the directory is specified as a named file, except that it silently ignores non-SCCS and unreadable files.

If you supply a dash (-) in place of a filename, **comb** reads the standard input. It takes each line of the input to be the name of an SCCS file to be processed, silently ignoring non-SCCS and unreadable files. **Comb** writes the generated Shell procedure on the standard output.

The valid options are described below. Each is explained as though only one named file is to be processed, but the effects of any option apply independently to each named file. If no options are specified, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

OPTIONS

- clist** Specify a *list* of deltas to be preserved. See **get(1)** for the syntax of a *list*.
- o** Access the reconstructed file at the release of the delta to be created. This activity occurs for each **-e** generated. By default, the reconstructed file is accessed at the most recent ancestor. Using this option may decrease the size of the reconstructed SCCS file and alter the shape of the delta tree of the original file.
- pSID** Specify the SCCS Identification string (SID) of the oldest delta to be preserved. Discard all older deltas in the reconstructed file.
- s** Generate a Shell procedure which, when run, produces a report on each file. This report will contain: the filename, size (in blocks) after combining, original size (in blocks), and percentage change computed by the following:

$100 * (\text{original} - \text{combined}) / \text{original}$

Use this option before any SCCS files are actually combined, to determine how much space is saved by the combining process.

CAUTIONS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, the reconstructed file may actually be larger than the original.

FILES

	reconstructed SCCS file
<i>comb?????</i>	temporary

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

admin(1), **delta(1)**, **get(1)**, **help(1)**, **prs(1)**, **sh(1)**, **scsfile(4)**.

NAME

comm – select or reject lines common to two sorted files

USAGE

comm [- [123]] *file1 file2*

DESCRIPTION

Comm reads *file1* and *file2*, producing a three-column output showing lines only in *file1*, lines only in *file2*, and lines in both files. Both files being read should be ordered in ASCII collating sequence. Refer to **sort(1)** for more information about this sequence.

If you specify a dash (-) in place of a filename, **comm** reads the standard input. Flags 1, 2, or 3 suppress printing of the corresponding column. Using all three flags at once creates a no-op.

EXAMPLES

To print only the lines common to both *file1* and *file2*, type the following:

```
# comm -12 file1 file2
```

To suppress columns 2 and 3, and then print lines in *file 1* (but not in *file2*), use this command:

```
# comm -23 file1 file2
```

RELATED INFORMATION

cmp(1), **diff(1)**, **sort(1)**, **uniq(1)**.

NAME

cp – copy files

USAGE

cp *file1* [*file2* ...] *target*

DESCRIPTION

Use **cp** to copy *file(s)* to a specified *target*. Under no circumstances can any of the *files* being manipulated be the same as the *target*, so take care when using shell metacharacters. If *target* is a directory, *file(s)* will be copied to that directory. If *target* is a file, its old contents are replaced by the contents of *file*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file. If *target* is not a file, **cp** creates a new file with the same mode as *file1*. The owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, or group. **Cp** sets the last modification time of *target* (and last access time, if *target* did not exist). It also sets the last access time of *file1* to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

RELATED INFORMATION

chmod(1)

cpio(1)

rm(1)

NAME

cpio – copy file archives in and out

USAGE

cpio -o [*acBv*]

cpio -i [*BcdmrtuvfsSb6*] [*patterns*]

cpio -p [*adlmruv*] *directory*

DESCRIPTION

Cpio -o (copy out) reads the standard input to obtain a list of pathnames. It then copies the files found onto the standard output, together with pathname and status information. Output is padded to a 512-byte boundary.

Cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, the question mark (?), asterisk (*), and [...] match the slash (/) character. You may specify multiple *patterns*. If you do not specify any *patterns*, the default for *patterns* becomes an asterisk (i.e., meaning to select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files are those of the previous **cpio -o**. The owner and group of the files are that of the current user (unless the user is super-user, which causes **cpio** to retain the owner and group of the files of the previous **cpio -o**).

Cpio -p (pass) reads the standard input to obtain a list of file pathnames that are conditionally created and copied into the destination *directory* tree based on the keyletter arguments described below.

KEYLETTER ARGUMENTS

- | | |
|---|--|
| a | Reset the access times of input files after they have been copied. |
| c | Write <i>header</i> information in ASCII character form for portability. |
| d | Create <i>directories</i> as needed. |
| r | Interactively <i>rename</i> files. If a null line is input, skip the file. |
| t | Print a table of contents of the input, but create no files. |
| u | Copy unconditionally (e.g., an older file can replace a newer file with the same name). |
| v | Print a verbose list of filenames. When used with the <i>t</i> keyletter argument described above, the table of contents generated looks like the output of the <i>ls</i> (1) command used with the <i>-l</i> option (i.e., <i>ls-l</i>). |
| l | Whenever possible, link files rather than copying them. This option is |

- only usable with the **-p** (pass) option.
- m** Retain previous file modification time. Ineffective on directories being copied.
- f** Copy in all files except those in *patterns*.
- s** Swap bytes. Use only with the **-i** (copy in) option.
- S** Swap halfwords. Use only with the **-i** (copy in) option.
- b** Swap both bytes and halfwords. Use only with the **-i** (copy in) option.
- 6** Process an old (e.g., UNIX System *Sixth* Edition format) file. This option is Only useful with the **-i** (copy in) option.
- h** Swap bytes in header information for VAX to 68000 and vice-versa (i.e., writing on one type of machine and reading on a different type of machine).

EXAMPLE

To copy the contents of a directory into an archive, use the following command:

```
ls | cpio -o >/dev/mt/0m
```

CAUTIONS

Pathnames are restricted to 128 characters. If too many unique linked files exist, the program runs out of memory to keep track of them and, thereafter, loses linking information.

Only the super-user can copy special files.

RELATED INFORMATION

ar(1), find(1), ls(1), sh(1), cpio(4).

NAME

cpp – the C language preprocessor

USAGE

/usr/lib/cpp [*option ...*] [*ifile* [*ofile*]]

DESCRIPTION

Cpp is a C language preprocessor. It is invoked by **lint(1)**, and is also available for general use. On DOMAIN/IX Systems, it plays no part in normal C program compilation.

The DOMAIN C compiler (**/com/cc**) has its own C preprocessor. Refer to the *DOMAIN C Language Reference* for more information.

Cpp optionally accepts two filenames as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

Note: When newline characters were found in argument lists for macros to be expanded, previous versions of **cpp** produced the new-lines as they were found and expanded. The current version of the command replaces these newlines with blanks to alleviate problems that previous versions had when this occurred.

Cpp understands two special names. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by **cpp**, and **__FILE__** is defined as the current filename (as a C string) as known by **cpp**. They can be used anywhere (including macros) just as any other defined name.

OPTIONS

- P** Preprocess the input without producing the line control information used by the next pass of the C compiler.
- C** Do not strip C-style comments (except those found on **cpp** directive lines).
- Uname** Remove any initial definition of *name*, where *name* is a reserved symbol predefined by the particular preprocessor. The current list of these possibly reserved symbols includes the following:
 - operating system: **ibm**, **gcos**, **os**, **tss**, **unix**, **aegis**
 - hardware: **interdata**, **pdp11**, **u370**, **u3b**, **u3b5**, **vax**, **apollo**
 - UNIX system variant: **RES**, **RT**
 - lint(1)**: **lint**
- Dname**
- Dname=def** Define *name* as if by a **#define** directive. If no **=def** is given, define *name* as one (1). This option has lower precedence than the **-U** option.

That is, if the same name is used in both a `-U` option and a `-D` option, the name will be undefined regardless of the order of the options.

- `-T` Use only the first eight characters for distinguishing different preprocessor names. This behavior is the same as previous preprocessors with respect to the length of names. Included for backward compatibility.
- `-Idir` Change the algorithm used to search for `#include` files with names not preceded by a slash (/) to look in *dir* before looking in the directories on the standard list. Thus, `#include` files with names enclosed in double quotes (" ") will be searched for first in the directory of the file with the `#include` line, then in directories named in `-I` options, and last in directories on a standard list. For `#include` with names enclosed in `<>`, the directory of the file with the `#include` line is not searched.

DIRECTIVES

All `cpp` directives start with lines beginning with a pound sign (#). Any number of blanks and tabs are allowed between the pound sign and the directive. The test directives and the possible `#else` directives can be nested.

The valid directives are listed below. In addition to these, the user may specify the following DOMAIN `cpp` directives: `#attribute`, `#debug`, `#eject`, `#nolist`, `#list`, `#module`, `#section`, and `#systype`. For a description of each of these, see the current revision of the *DOMAIN C Language Reference*.

`#define name token-string`

Replace subsequent instances of *name* with *token-string*.

`#define name(arg, ..., arg) token-string`

Note that there can be no space between *name* and the left parenthesis.

Replace subsequent instances of *name* followed by a left parenthesis, a list of comma-separated set of tokens, and a right parenthesis by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, `cpp` restarts its scan for names to expand at the beginning of the newly-created *token-string*.

`#undef name`

Forget the definition of *name* (if any) from now on.

`#include "filename"`

`#include <filename>`

Include, at this point, the contents of *filename* (which will then be run through

cpp). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See the **-I** option above for more detail.

#line *integer-constant* "*filename*"

Generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file from which it is derived. If "*filename*" is not given, the current filename is unchanged.

#endif

End a section of lines started by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*

Include the lines following in the output only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*

Do not include the lines following in the output only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*

Include the lines following in the output only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary **-**, **!**, and **~** operators are legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined (name)** or **defined name**. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only those operators, integer constants, and names known by **cpp** should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else Reverse the notion of the test directive that matches this directive. If lines previous to this directive are ignored, the following lines will appear in the output (and vice versa).

FILES

/usr/include standard directory for **#include** files

DIAGNOSTICS

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

RELATED INFORMATION

cc(1), **m4(1)**.

NAME

crontab - make a user crontab file

USAGE

crontab *option*
crontab [*file*]

DESCRIPTION

Crontab copies the specified *file* (or the standard input if no *file* is named), into a directory that holds all users' *crontab* files.

You can use **crontab** if your name appears in the file */usr/lib/cron/cron.allow*. If that file does not exist, the file */usr/lib/cron/cron.deny* determines whether you are allowed to use **cron**. If */usr/lib/cron/cron.allow* does not exist and */usr/lib/cron/cron.deny* is empty, global use is allowed. If neither file exists, only *root* can submit a job. The allow/deny files consist of one user name per line.

OPTIONS

Crontab recognized the following options:

- l List the current user's *crontab* file.
- r Remove the current user's *crontab* file.

CRONTAB FILE FORMAT

A *crontab* file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following information:

minute	(0-59)
hour	(0-23)
day of month	(1-31)
month of year	(1-12)
day of week	(0-6, 0=Sunday)

Each of these patterns may be either an asterisk (*) to signify all legal values, or a list of elements separated by commas. An element is either a number, or two numbers separated by a dash (-) to indicate an inclusive range. Note that you can specify days by using day of the month and/or day of the week. If you use both fields as a list of elements, both are adhered to. For example,

0 0 1,15 * 1 *command*

runs *command* on the first and fifteenth of each month, as well as on every Monday. To specify the days in a single field, the other field should be set to an asterisk (*).

For example:

```
0 0 * * 1 command
```

runs *command* only on Mondays.

The sixth field of a line in a *crontab* file is a string that is executed by the shell at the specified times. A percent character (%) in this field, unless escaped by a backslash (\) is translated to a newline character. Only the first line of the command field, up to a percent (%) or end of line, is executed by the shell. The other lines are made available to the command as standard input.

Cron invokes the shell from your \$HOME directory with an *arg0* of sh(1). Your *profile* will not be executed unless you make an explicit request in the *crontab* file. The **crontab(1M)** command supplies a default environment for every shell, defining HOME, LOGNAME, SHELL (= /bin/sh), and PATH (= /bin:/usr/bin:/usr/sbin).

EXAMPLES

The first line below installs the file */tmp/ct* as your *crontab*. The second line lists that file.

```
# crontab /tmp/ct
# crontab -l
00 * * * * echo 'date' >>/tmp/cron
```

The *crontab* file listed above writes the date into a file once every hour.

CAUTIONS

If you do not redirect standard output and standard error when using **crontab**, any generated output or errors are mailed to you.

FILES

<i>/usr/lib/cron</i>	main cron directory (link to ' <i>node_data/cron</i>)
<i>/usr/spool/cron/crontabs</i>	spool area
<i>/usr/lib/cron/log</i>	accounting information
<i>/usr/lib/cron/cron.allow</i>	list of allowed users
<i>/usr/lib/cron/cron.deny</i>	list of denied users

RELATED INFORMATION
sh(1), cron(1M).

NAME

csh – a shell (command interpreter) with C-like syntax

USAGE

csh [*options*] [*arg* ...]

DESCRIPTION

Csh is a command language interpreter that uses a history mechanism, job control facilities, and a C-like syntax.

It begins by executing commands from the *.cshrc* file in your *home* directory. If this is a log-in shell, then it also executes commands from your *.login* file. Frequently, CRT users place an *stty crt* command in their *.login* file, and also invoke *tset(1)* there.

Normally, the shell then begins reading commands from the terminal, prompting with a percent sign (%). Upon reading a line of command input, the shell breaks it into *words*, places this sequence of words on the command history list, parses it, and then executes each command in the current line.

When a log-in shell terminates, it executes commands from the *.logout* file in your home directory.

LEXICAL STRUCTURE

Usually, *csh* splits input lines into words at blanks and tabs. The following, however, are exceptions to this:

1. An ampersand (&), pipe (|), semicolon (;), greater-than (>), less-than (<), or parenthetical character forms separate words. If these characters are doubled in pairs, the pairs form single words. You can make these parser metacharacters a part of other words or prevent their special meaning by preceding them with a backslash (\). A newline preceded by a backslash is equivalent to a blank.
2. Strings enclosed in matched pairs of quotations, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. Within pairs of single or double quotations, a newline preceded by a backslash gives a true newline character.
3. When the shell's input is not a terminal, a pound sign (#) introduces a comment that continues to the end of the input line. To prevent this special meaning, you may precede the comment by a backslash and place it in single quotation marks, or place it in double quotation marks.

COMMANDS

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by pipe (|) characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. You can separate a series of pipelines by a semicolon

(;) to signal the sequential execution of commands. If you follow this sequence with an ampersand (&), `cs`h processes these pipelines in the background.

Any of the above may be placed in parentheses to form a simple command (which may be a component of a pipeline, etc.) You can also separate pipelines with double pipe characters (||) or double ampersands (&&) to indicate, as in the C language, that the second is to be executed only if the first fails or succeeds respectively.

JOBS

Csh associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started asynchronously with an ampersand (&), csh prints a line similar to the following:

```
[1] 1234
```

This indicates that the job, which was started asynchronously, was job number 1 and had one (top-level) process whose process ID was 1234.

To stop a job, you must execute an interrupt (usually a control key bound by the Display Manager). Once csh has indicated that the job has been stopped (by printing a prompt), you can manipulate the state of this job. You may put it in the background with the *bg* command, or run some other commands and then eventually bring the job back into the foreground with the foreground command, *fg*. A suspend (normally a \uparrow Z) takes effect immediately, causing csh to discard pending output and unread input. Note that, to set the suspend character in a pad, you must give the following command from the Display Manager: `kd ^Z dq-c 120028 ke`. On a CRT, use the `stty(1)` command.

A job being run in the background stops if it tries to read from the terminal. Background jobs are normally allowed to produce output, but you can disable this by giving the `stty tostop` command. Setting this TTY option causes background jobs to stop when they try to produce output in the same manner as they do when they read input.

There are several ways to refer to jobs in the shell. A percent sign (%) introduces a job name. Job number 1, for example, becomes *%1*. Naming a job brings it to the foreground; thus, *%1* is a synonym for *fg %1*, bringing job 1 back into the foreground. Similarly, specifying *%1 &* resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous. Therefore, *%ex* normally restarts a suspended *ex(1)* job, if there is only one suspended job whose name began with the string *ex*. It is also possible to specify *%?string* to indicate a job whose text contains *string*, if there is only one such job.

Csh maintains a notion of the current and previous jobs. In output pertaining to jobs, it marks the current job with a plus sign (+), and the previous job with a minus sign (-). A percent and a plus sign (%+) refers to the current job, and a percent and minus sign (%-) refers to the previous job. For close analogy with the syntax of the *history* mechanism (described below), a double percent sign (%%) also represents the current

job.

STATUS REPORTING

Csh knows immediately when the state of a process changes. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. If, however, you set the **notify** shell variable, csh immediately reports status changes in background jobs. The **notify** shell command also marks a single process so that its status changes are immediately reported. By default, **notify** marks the current process. Thus, you only have to type **notify** after starting a background job to mark it.

If you attempt to leave the shell while jobs are stopped, a warning message will appear. The **jobs** command allows you to see which jobs are affected. A second attempt to exit causes the suspended jobs to terminate without warning.

HISTORY SUBSTITUTIONS

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and much confidence. History substitutions begin with an exclamation point (!) and may start anywhere in the input stream (providing that they do not nest). Precede the exclamation point with a backslash (\) to prevent its special meaning. For convenience, the character is passed unchanged when it is followed by a blank, tab, newline, equal sign, or left parenthesis. History substitutions also occur when an input line begins with a caret (^). Before being executed, input lines containing history substitution are echoed on the terminal as they could have been typed without history substitution.

Csh saves input commands consisting of one or more words on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream, the size of which is controlled by the **history** variable. The previous command is always retained, regardless of its value. Commands are numbered sequentially from one (1).

As an example, consider the following output from the **history** command:

```
9  write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an exclamation point (!) in the prompt string.

With the current event 13, you can refer to previous events by event number, as in !11 for event 11; relatively, as in !-2 for event 11; by a prefix of a command word, as in !d for event 12 or !wri for event 9; or by a string contained in a word in the command, as in !?mic? also referring to event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, double exclamation points (!!) refer to the previous command. Thus, !! alone is essentially another way of performing what the redo shell command usually does.

To select words from an event, you can follow the event specification by a colon (:) and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are as follows:

- 0 First (command) word
- n Nth argument
- ^ First argument, i.e., 1
- \$ Last argument
- % Word matched by (immediately preceding) ?s? search
- x-y Range of words
- y Abbreviation of 0-y
- * Abbreviation of ^-\$ (or nothing if only 1 word in event)
- x* Abbreviation of x-\$
- x- Like x*, but omitting word \$

You can omit the colon (:) separating the event specification from the word designator, if the argument selector begins with a caret (^), dollar sign (\$), asterisk (*), or percent (%). Furthermore, you can place a sequence of modifiers after the optional word designator, preceding each modifier with a colon (:).

Csh defines the following modifiers:

- h Remove a trailing pathname component, leaving the head
- r Remove a trailing .xxx component, leaving the root name
- e Remove all but the extension .xxx part
- s/l/r/ Substitute l for r
- t Remove all leading pathname components, leaving the tail

<code>&</code>	Repeat the previous substitution
<code>g</code>	Apply the change globally, prefixing the above, e.g., <code>g&</code>
<code>p</code>	Print the new command but do not execute it
<code>q</code>	Quote the substituted words, preventing further substitutions
<code>x</code>	Like <code>q</code> , but break into words at blanks, tabs, and newlines

Unless preceded by a `g`, the modification is applied only to the first modifiable word. With substitutions, every word should be applicable or an error occurs.

The left-hand side of substitutions are not regular expressions in the sense of the editors, but rather they are strings. Any character may be used as the delimiter in place of a regular slash mark (/). A backslash (\) quotes the delimiter into the `l` and `r` strings. The ampersand character (&) in the right-hand side is replaced by the text from the left. A backslash also quotes an ampersand (&). A null `l` uses the previous string either from an `l` or from a contextual scan string `s` in `!s?`. The trailing delimiter in the substitution, as well as the trailing question mark (?) in a contextual scan, may be omitted if a newline follows immediately.

A history reference can be given without an event specification, e.g., `!$`. In this case, the reference is to the previous command unless a previous history reference occurred on the same line (in which case this form repeats the previous reference). Thus, `!foo?^ !$` gives the first and last arguments from the command matching `?foo?`.

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a caret (^). This is equivalent to `!:s^` and provides a convenient shorthand for substitutions on the text of the previous line. Thus, `^lb^lib` fixes the spelling of *lib* in the previous command. Finally, a history substitution may be surrounded with braces ({ }) to insulate it from the characters that follow. Thus, after `ls -ld ~paul` we might do `!{l}a` to do `ls -ld ~paula` while `!la` would look for a command starting with *la*.

QUOTATIONS WITH SINGLE AND DOUBLE QUOTES

Placing strings in single and double quotes prevents all or some of the remaining substitutions. Those enclosed in single quotes are prevented any further interpretation; those in double quotes may be expanded as described below.

In both cases, the resulting text becomes all or part of a single word. In only one special case (see *COMMAND SUBSTITUTION* below) does a double-quoted string yield parts of more than one word; single-quoted strings never do.

ALIAS SUBSTITUTION

Csh maintains a list of aliases that can be established, displayed, and modified by the `alias` and `unalias` commands. After it scans a command line, the shell parses the line into distinct commands and checks the first word of each command, left-to-right, for an alias. If it finds one, csh rereads the text that is the alias for that command (with the history mechanism available) as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

For example, if the alias for `ls` is `ls -l`, the command `ls /usr` would map to `ls -l /usr`, the argument list here being undisturbed. Similarly, if the alias for `lookup` was `grep !^ /etc/passwd`, then `lookup bill` would map to `grep bill /etc/passwd`.

Every time the shell finds an alias, it transforms the input text and begins the aliasing process again on the reformed input line. Looping is prevented (if the first word of the new text is the same as the old) by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus, you can `alias print 'pr\'* | lpr'` to make a command that does a `pr(1)` on its arguments to the line printer.

VARIABLE SUBSTITUTION

Csh maintains a set of variables, each having as value a list of zero or more words. The shell sets some of these variables, and merely refers to others. For instance, the `argv` variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

You may display and change the values of variables by using the `set` and `unset` commands. Of the variables referred to by the shell, a number are toggles. The shell does not care what their value is, only whether they are set or not. For instance, the `verbose` variable is a toggle that causes command input to be echoed. The setting of this variable results from the `-v` command line option.

Other operations treat variables numerically. The command represented by the `@` sign (`@`) permits numeric calculations to be performed, with the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. In numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After csh has aliased and parsed the input line, and before executing each command, it performs variable substitution keyed by dollar sign (\$) characters. This expansion can be prevented by preceding the dollar sign (\$) with a backslash (\), except within double quotes ("), where it *always* occurs, and within single quotes (') where it *never* occurs. Strings quoted by a single quote are interpreted later (see *COMMAND SUBSTITUTION* below), so the dollar sign (\$) substitution does not occur there until later, if at

all. A dollar sign (\$) is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. Therefore, the first (command) word to this point can generate more than one word; the first word becomes the command name, and the rest become arguments.

Unless enclosed in double quotes or given the :q modifier, the results of variable substitution may eventually be command and filename substituted. Within double quotes, a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variables value separated by blanks. When the :q modifier is applied to a substitution, the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided to introduce variable values into the shell input. Except as noted, you cannot reference a variable that is not set. You may apply the :h,:t,:r, and :gr modifiers to most of the substitutions below. Ones that you cannot do this with are marked accordingly. If braces appear in the command form, then the modifiers must appear within the braces. Only one modifier beginning with a colon (:) can be applied on each expansion preceded by a dollar sign (\$).

\$name

\${name}

Replace text by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters, which would otherwise be part of it. shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character (_) is considered a letter. If *name* is not a shell variable, but is set in the environment, then that value is returned. However, colon (:) modifiers and the other forms given below are not available in this case.

\$name[selector]

\${name[selector]}

Select only some of the words from the value of *name*. The selector is subjected to dollar sign (\$) substitution and may consist of a single number or two numbers separated by a dash (-). The first word of a variable's value is numbered 1. If the first number of a range is omitted, it defaults to 1. If the last member of a range is omitted, it defaults to \$#*name*. The selector asterisk (*) selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

\$#name

\${#name}

Give the number of words in the variable. This is useful for later use in a [selector].

<code>\$0</code>	Substitute the name of the file from which command input is being read. An error occurs if the name is not known.
<code>\$number</code> <code>\${number}</code>	Equivalent to <code>\$argv[number]</code> .
<code>\$*</code>	Equivalent to <code>\$argv[*]</code> .
<code> \$?name</code> <code>\${ ?name}</code>	Substitute the string 1 if name is set, 0 if it is not. This substitution may not be modified with modifiers preceded by a colon (:).
<code> \$?0</code>	Substitute 1 if the current input filename is known, 0 if it is not. This substitution may not be modified with modifiers preceded by a colon (:).
<code> \$\$</code>	Substitute the decimal process number of the parent shell. This substitution may not be modified with modifiers preceded by a colon (:).
<code> \$<</code>	Substitute a line from the standard input, with no further interpretation. Useful for reading from the keyboard in a shell script. This substitution may not be modified with modifiers preceded by a colon (:).

COMMAND SUBSTITUTION

Csh applies the remaining substitutions, command and filename substitution, selectively to the arguments of built-in commands. This means that portions of expressions not evaluated are not subjected to these expansions. Names for commands that are not internal to the shell are substituted separately from the argument list. This occurs very late, after input/output redirection is performed, and in a child of the main shell.

Enclosing a command in backquotes indicates command substitution. Csh usually breaks the output from such a command into separate words at blanks, tabs, and newlines. It discards null words, and uses the modified text to replace the original string. Within double quotes, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

If a word contains an asterisk (*), question mark (?), left bracket ([), or left brace ({), or it begins with a tilde (~), then that word is a candidate for filename substitution, also known as "globbing." Csh regards the word as a pattern, replacing it with an alphabetically sorted list of filenames that match the pattern. In a list of words specifying filename substitution, at least one pattern must match an existing filename, but each pattern need not match. Only an asterisk (*), question mark (?), and left bracket

(I) imply pattern matching. The tilde (~) and left brace (()) are like abbreviations.

FILE SUBSTITUTION

In matching filenames, you must match a period (.) at the beginning of a filename or immediately following a regular slash (/) explicitly. This is also true for the slash itself. An asterisk (*) matches any string of characters, including the null string. A question mark (?) matches any single character. The sequence [...] matches any one of the characters enclosed. Within such a sequence, a pair of characters separated by a backslash (\) matches any character lexically between the two.

A tilde (~) at the beginning of a filename refers to home directories. Standing alone, it expands to your home directory (reflected in the value of the variable *home*). When followed by a name consisting of letters, digits, and dashes (-), *cs*h searches for a user with that name and substitutes his or her home directory. Thus, *~ken* might expand to */usr/ken* and *~ken/chmach* to */usr/ken/chmach*. If a tilde is followed by a character other than a letter or slash (/), or if it appears somewhere other than at the beginning of a word, the shell leaves it undisturbed.

The metanotation *a{b,c,d}e* is a shorthand for *abe ace ade*. Left-to-right order is preserved. The results of matching are sorted separately at a low level to preserve this order (nesting is acceptable). Thus, *~source/sl/{oldls,ls}.c* expands to */usr/source/sl/oldls.c /usr/source/sl/ls.c* whether or not these files exist without any chance of error if the home directory for *source* is */usr/source*. Similarly, *../{memo,*box}* might expand to *../memo ../box ../mbox*. (Note that *memo* was not sorted with the results of matching **box*.) As a special case, the shell passes all single unmatched braces or an empty pair of braces undisturbed.

INPUT/OUTPUT

To redirect the standard input and standard output of a command, use the following syntax:

- < name* Open the file *name* (which is first variable-, command-, and filename-expanded) as the standard input.
- << word* Read the shell input up to a line identical to *word*. *Word* is not variable-, filename-, or command-substituted. Each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting backslash (\), double quotation ("), or backquote (`) character appears in *word*, *cs*h performs variable and command substitution on the intervening lines, allowing the backslash to quote a dollar sign (\$), a backslash (\), and backquotes (`). Commands that are substituted have all blanks and tabs preserved. All newlines except for the final one are also preserved. The resulting text is placed in an anonymous temporary file, which is given to the command as standard input.

> *name*
>! *name*
>& *name*
>&! *name*

Use the file *name* as standard output. If the file does not exist, create it; if the file does exist, truncate it, discarding its previous contents.

If the variable **noclobber** is set, then the file must not exist, or it must be a character special file (e.g., a terminal or */dev/null*), or an error results. This helps prevent accidental destruction of files. An exclamation point (!) suppresses this check.

Forms involving an ampersand (&) route the diagnostic output into the specified file, as well as the standard output. *Name* is expanded in the same way as input filenames beginning with a less-than character (<) are.

>> *name*
>>& *name*
>>! *name*
>>&! *name*

Use the file *name* as standard output, but place output at the end of the file. If the variable **noclobber** is set, then it is an error for the file not to exist unless one of the forms beginning with an exclamation point (!) is given.

A command receives the environment in which the shell was invoked as modified by the input/output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather, they receive the original standard input of the shell. The mechanism identified with doubled less-than characters (<<) should be used to present in-line data. This permits shell command scripts to function as components of pipelines and allows the shell to block-read its input. Note that the default standard input for a command run detached is *not* modified to be the empty file */dev/null*. Rather, the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and you are notified (see *JOBS* above.)

Diagnostic output may be directed through a pipe with the standard output. Simply use an ampersand (&) after the pipe (|) to do this.

EXPRESSIONS

A number of the built-in commands take expressions that have operators similar to those used for C language, with the same precedence. These expressions appear in the @, exit, if, and while commands. The following operators are available:


```

|| && |  & == != =~ !~

<= >= < > << >> + - * / % ! ~ ( )

```

Here the precedence increases to the right, The following characters are, in groups, at the same level:

```

==  !=  =~  !~

<=  >=  <  >

<<  >>

+    -

*    /    %

```

The following operators compare their arguments as strings:

```

==  !=  =~  !~

```

All others operate on numbers. The operators `=~` and `!~` are like `==` and `!=` except that the right-hand side is a *pattern* (containing, for example, asterisks, question marks, and instances of [...] characters) against which the left-hand operand is matched. This reduces the need for using the `switch` statement in shell scripts when all that is really needed is pattern-matching.

Csh considers strings beginning with a zero to be octal numbers. It interprets null or missing arguments as zero. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word. You should surround them by spaces, except when they are adjacent to components of expressions that are syntactically significant to the parser (e.g., ampersands, pipe characters, etc.).

Also available in expressions as primitive operands are command executions enclosed in braces (`{` and `}`), and file enquiries of the form `-l name` where *l* is one of the following:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size

f plain file
d directory

Csh performs command and filename expansion on the specified name, and then checks to see if it has the specified relationship to the real user. If the file does not exist, or if it is inaccessible, then all inquiries return false (0). Command executions succeed, returning true (1), if the command exits with status 0; otherwise, they fail, returning false (0). If you require more detailed status information, execute the command outside an expression and examine the status variable.

CONTROL FLOW

Csh contains a number of commands to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The `foreach`, `switch`, and `while` statements, as well as the `if-then-else` form of the `if` statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward `gotos` succeed on nonseekable inputs.)

BUILT-IN COMMANDS

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, then it is executed in a sub-shell.

alias Print all aliases.

alias *name*

Print the alias for *name*.

alias *name wordlist*

Assign the specified *wordlist* as the alias of *name*. The *wordlist* is command- and filename-substituted. *Name* is not allowed to be *alias* or *unalias*.

alloc Show the amount of dynamic core in use, broken down into used and free core, and the address of the last location in the heap. With an argument, this command shows each used and free block on the internal dynamic memory chain, indicating its address, size, and status (used or free). This is a debugging command and may not work in production versions of the shell. It requires a modified version of the system memory allocator.

bg

bg *%job ...*

Put the current or specified jobs into the background, continuing them if they were stopped.

break Resume execution after the end of the nearest enclosing **foreach** or **while**. Execute the remaining commands on the current line. Multi-level breaks are thus possible by writing them all on one line.

breaksw

Break from a **switch**, resuming after the **endsw**.

case label:

Specify a label in a **switch** statement.

cd

cd name

chdir

chdir name

Change the shell's working directory to directory *name*. If no argument is given, change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory and does not begin with a slash (/), or a slash preceded by one or two periods (./ or ../), check each component of the variable **cdpath** to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with a slash, check to see if it is a directory.

continue

Continue execution of the nearest enclosing **while** or **foreach**. Execute remaining commands on the current line.

default:

Label the default case in a **switch** statement. This command should follow all case labels.

dirs Print the directory stack. The top of the stack is at the left, and the first directory in the stack is the current directory.

echo wordlist

echo -n wordlist

Write the specified words to the shell's standard output, separated by spaces, and terminated with a newline (unless the **-n** option is specified).

else

end

endif

endsw See the description of the **foreach**, **if**, **switch**, and **while** statements below.

eval arg ...

Read the arguments as input to the shell, executing the resulting command(s) in the context of the current shell. This occurs as in `sh(1)`. The command is generally used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See `tset(1)` for an example of using `eval`.

exec *command*

Execute the specified command in place of the current shell.

exit Exit with the value of the status variable.

exit (*expr*)

Exit with the value of the specified *expr*.

fg

fg %*job* ...

Bring the current or specified jobs into the foreground, continuing them if they were stopped.

foreach *name* (*wordlist*)

...
end Successively set the variable *name* to each member of *wordlist*, and execute the sequence of commands between this command and the matching *end*. (Both **foreach** and **end** must appear alone on separate lines.)

Use the **continue** command to continue the loop prematurely. Use the **break** command to terminate it prematurely. When the shell reads this command from the terminal, it reads the loop once, prompting with a question mark (?) before executing any statements in the loop. If you make a mistake typing in a loop at the terminal, you can interrupt it by typing **dq -i**.

glob *wordlist*

Perform the same function as the **echo** command, but do not recognize backslash escapes, and delimit words by null characters in the output. Use this command with programs that use the shell to filename-expand a list of words.

gotoword

Perform filename- and command-expansion on the specified *word* to yield a string of the form *label*. Cause the shell to rewind input as much as possible and search for a line of the form *label*: (possibly preceded by blanks or tabs). Continue execution after the specified line.

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands and avoiding instances of the **exec** command. An **exec** is attempted for each component of the *path* where the hash function indicates a

possible hit, and in each component that does not begin with a slash.

history

Display the history event list.

history *n*

Print only the *n* most recent events in the history event list.

history -r *n*

Print the most recent (rather than the oldest) events in the history event list.

history -h *n*

Print the history event list without leading numbers. Use this command to produce files suitable for sourcing with the -h option to the source built-in command.

if (*expr*) *command*

If the specified expression evaluates true, then execute the single *command* with arguments. Variable substitution on *command* happens early, at the same time it does for the rest of the if command. The *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list.

Input/output redirection occurs even if *expr* is false, when command is *not* executed.

if (*expr*) then

...

else if (*expr2*) then

...

else

...

endif If the specified *expr* is true, then execute the commands to the first else; if else if *expr2* is true, then execute the commands to the second else, etc. Any number of else-if pairs are possible; only one endif is needed. The else part is optional. (The words else and endif must appear at the beginning of input lines; the if must appear alone on its input line or after an else.)

inlib *pathname*

Install a user-supplied library specified by *pathname* at the current shell program level. The library remains installed until the shell that installed it exits. User-inlibed libraries are only available to programs that are run in-process. Look under the description of in-process execution given in the predefined variable section of this manual entry. Also refer to the description of the DOMAIN command /com/lib in the *DOMAIN System Command Reference*.

jobs List the active jobs.

jobs -l

List the active jobs, but also include process IDs.

kill %job**kill -sig %job ...****kill pid****kill -sig pid ...**

kill -l Send either the TERM (terminate) signal or the specified signal to the jobs or processes indicated. Provide signals by number or by names (as given in */usr/include/signal.h*, stripped of the *SIG* prefix. A **kill -l** lists the signal names. There is no default for this command. A **kill** alone on the command line does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

login Terminate a log-in shell, replacing it with an instance of */bin/login*. This is one way to log off, and it is included for compatibility with *sh(1)*.

logout Terminate a log-in shell. This command is especially useful if *ignoreeof* is set.

nice Set the *nice(1)* increment for this shell to 4.

nice +number

Set the *nice(1)* increment to the given positive *number*.

nice command

Run *command* at *nice(1)* priority 4.

nice +number command

Run *command* at positive *number* *nice(1)* priority. The command is always executed in a sub-shell, and the restrictions placed on commands in simple *if* statements apply.

nohup When used in a shell script, ignore hangups for the remainder of the script.

nohup command

Run the specified *command* with hangups ignored. This happens to all processes detached with an ampersand (&).

notify**notify %job ...**

Notify the user asynchronously when the status of the current or specified jobs changes (normally, notification is presented before a prompt). This is automatic, if the shell variable *notify* is set.

onintr Restore the default action of the shell on interrupts (to terminate shell scripts or to return to the terminal command input level). In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no

meaning, and interrupts continue to be ignored by the shell and all invoked commands.

onintr -

Ignore all interrupts.

onintr*label*

Execute a *gotolabel* when an interrupt is received or a child process terminates because it was interrupted.

popd Pop the directory stack, returning to the new top directory. The elements of the directory stack are numbered from zero, starting at the top.

popd +n

Discard the *n*th entry in the directory stack.

pushd Exchange the top two elements of a directory stack.

pushd *name*

Change to *name* directory and push the old current working directory onto the directory stack.

pushd +n

Rotate the *n*th argument of the directory stack around to be the top element and change to it. The members of the directory stack are numbered from zero, starting at the top.

rehash

Recompute the internal hash table of the contents of the directories in the *path* variable. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if someone changes the contents of one of the system directories.

repeat *count command*

Execute the specified *command* (subject to the same restrictions as the *command* in the one-line if statement above) *count* times. I/O redirections occur exactly once, even if *count* is zero.

set Show the value of all shell variables. Variables which have other than a single word as their value print as a parenthesized word list.

set *name*

Set *name* to the null string. In all cases, the value is command- and filename-expanded.

set *name=word*

Set *name* to the single *word*. In all cases, the value is command- and filename-

expanded.

set *name*[*index*]=*word*

Set the *index*th component of *name* to *word*. This component must already exist. In all cases, the value is command- and filename-expanded.

set *name*=(*wordlist*)

Set *name* to the list of words in *wordlist*. In all cases, the value is command- and filename-expanded. These arguments may be repeated to set multiple values in a single set command. Note, however, that variable expansion happens for all arguments before any setting occurs.

setenv *name value*

Set the value of the environment variable *name* to be *value*, a single string. The most commonly used environment variables -- USER, TERM, and PATH -- are automatically imported to and exported from the *cs*h variables *user*, *term*, and *path*. You do not have to use *setenv* for these.

shift Shift the members of *argv* to the left, discarding *argv*[1]. It is an error for the *argv* variable not to be set or to have less than one word as its value.

shift *variable*

Shift the specified *variable* to the left.

source *name*

Read commands from *name*. You may nest *source* commands, but if you nest them too deeply, the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands.

source -h *name*

Place commands in the history list without executing them. Normally, input during *source* commands is not placed on the history list.

stop Stop the current job that is executing in the background.

stop %*job* ...

Stop the specified *job* that is executing in the background.

suspend

Cause the shell to stop immediately, much as if it had been sent a stop signal with ↑Z. This is most often used to stop shells started by *su*(1).

switch (*string*)

*case**str*1:

...

break*sw*

...

default:

...

breaksw

endsw Successively match each case label against the specified *string*, which is first command- and filename-expanded. The following metacharacters may be used in the case labels, which are variable-expanded: *, ?, and [...]. If none of the labels match before a default label is found, then begin the execution after the default label. Each case label and the default label must appear at the beginning of a line. The **breaksw** command causes execution to continue after the **endsw**. Otherwise, control may fall through case labels and default labels as in C programs. If no label matches and there is no default, execution continues after the **endsw**.

time Print a summary of time used by this shell and its children.

time *command*

Time the specified simple *command*, and print a time summary as described under the **time** variable. If necessary, create an extra shell to print the time statistic when the command completes.

umask

Display the file creation mask (in octal).

umask *value*

Set the file creation mask to the specified *value*. Common values for the mask are 002 (giving all access to the group and read and execute access to others) or 022 (giving all access except no write access for users in the group or others).

unalias *pattern*

Discard all aliases whose names match the specified *pattern*. Thus, all aliases are removed by **unalias** *. It is not an error for nothing to be **unaliased**.

unhash

Disable the internal hash table mechanism normally used to speed location of executed programs.

unset *pattern*

Remove all variables whose names match the specified *pattern*. Thus, all variables are removed by **unset** *. This has noticeably distasteful side-effects. It is not an error for nothing to be **unset**.

unsetenv *pattern*

Remove all variables whose names match the specified *pattern* from the environment. Also refer to the **setenv** built-in shell command above and the

printenv(1) command.

ver *systype*

With no arguments, return the current value of the SYSTYPE environment variable. With a *systype* argument, change the SYSTYPE environment variable to either *bsd4.2* or *sys5*, depending on which is specified.

wait Wait for all background jobs. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (*expr*)

...
end While the specified expression evaluates non-zero, evaluate the commands between the **while** and the matching **end**. You may use **break** and **continue** to terminate or continue the loop prematurely. (The **while** and **end** must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the **foreach** statement if the input is a terminal.

which Identify which file would have been executed had the command been submitted for execution. The command is submitted to normal alias and variable substitutions.

%job Bring the specified *job* number into the foreground.

%job &
Continue the specified *job* in the background.

@ Print the values of all the shell variables.

@ name = expr
Set the specified *name* to the value of *expr*. If the expression contains a greater-than (>), less-than (<), or pipe (|) character, then you must place at least that part within parentheses.

@ name[index] = expr
Assign the value of *expr* to the *indexth* argument of *name*. Both *name* and its *indexth* component must already exist. Operators are available as in C language. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* that would otherwise be single words. Special postfix double plus (++) and double minus (--) operators increment and decrement *name* respectively, i.e.,
@ i++.

PREDEFINED AND ENVIRONMENT VARIABLES

The following variables have special meaning to `cs`. Of these, `argv`, `cwd`, `home`, `path`, `prompt`, `Shell`, and `status` are always set by the shell. Except for `cwd` and `status`, this setting occurs only at initialization. These variables will not then be modified unless you explicitly perform the modification.

`Csh` copies the `USER` environment variable into the user variable, `TERM` into `term`, and `HOME` into `home`. It then copies these back into the environment whenever the normal shell variables are reset.

`Csh` handles the `PATH` environment variable in a similar manner. Do not worry about the setting for `PATH` other than in the file `.cshrc`. Inferior `csh` processes import the definition of `path` from the environment, and re-export it if you then change it.

argv	Set to the arguments to the shell. It is from this variable that positional parameters are substituted, i.e., <code>\$argv[1]</code> replaces <code>\$1</code> , etc.
cdpath	Give a list of alternate directories searched to find subdirectories in <code>cd</code> commands.
cwd	Give the full pathname of the current directory.
echo	Echo each command and its arguments just before the command is executed. This variable is set when the <code>-x</code> command line option is given. For non-built-in commands all expansions occur before echoing. Echo built-in commands before command and filename substitution, since these substitutions are then done selectively.
histchars	Change the characters used in history substitution, if a string value is specified. Use the first character of its value as the history substitution character, replacing the default exclamation point (!). The second character of its value replaces the <code>↑</code> character in quick substitutions.
history	Control the size of the history list. If a numeric value is specified, do not discard any command that has been referenced in that many events. Save the last executed command on the history list. The shell may run out of memory if the value of history is too large.
home	Represents the home directory of the invoker, initialized from the environment. The filename expansion caused by the presence of a tilde (<code>~</code>) character refers to this variable.
homedirchar	Change the character used to refer to the home directory variable, if a string value is specified. Unsetting this variable restores the tilde (<code>~</code>) as the default character.
ignoreeof	If set, ignore the end-of-file from terminal input devices. This

prevents shells from accidentally being killed by an EOF.

inprocess

If set, run new programs in the shell process (unless part of a pipe or running in background). When **inprocess** is set, programs run by the C shell can access user libraries installed via the **inlib** built-in command. Otherwise, the C shell always spawns a new process to run a new program. This variable is normally **unset** unless it has been set in the DM (by putting a line of the form “**env INPROCESS 'true'**” in the file *'node_data/startup.type'*), or unless the **-j** option is used with **cs**. Commands started in-process cannot be suspended or manipulated using the **cs** job-control facilities.

mail

Represent the files where the shell checks for mail. This is done after each command completion that results in a prompt, if a specified interval has elapsed. The shell will tell you that you have new mail, if the file exists with an access time not greater than its modify time. If the first word of the value of **mail** is numeric, it specifies a different mail-checking interval (in seconds) than the default (10 minutes). If you specify multiple mail files, the shell will tell you that you have new mail in *name*, when there is mail in the file *name*.

noclobber

Restrict output redirection to insure that files are not accidentally destroyed, and that redirections done with **>>** characters refer to existing files.

noglob

If set, inhibit filename expansion. Use this in shell scripts that do not deal with filenames, or after you have obtained a list of filenames and further expansions are not desirable.

nonomatch

If set, it is not an error for a filename expansion to not match any existing files; rather, the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e., **echo [** still gives an error.

notify

If set, notify the user asynchronously of job completions. By default, the shell presents job completions just before printing a prompt.

path

Use each word of the **path** variable to specify a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no **path** variable, then only full path-names execute. The usual search path is as follows:

. (period)
/bin

/usr/bin

This, however, may vary from system to system. For the super-user, the default search path is as follows:

/etc
/bin
/usr/bin

A shell that is given neither the `-c` nor the `-t` option will normally hash the contents of the directories in the `path` variable after reading `.cshrc`, and each time the `path` variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the `rehash` or the commands may not be found.

prompt

Read, from an interactive terminal input, the string printed before each command. If an exclamation point (!) appears in the string, replace it by the current event number (unless a preceding backslash is given). For the super-user, the default is a percent (%) or pound (#) sign.

savehist

Give a numeric value to control the number of history list entries saved in `~/.history` at log-out time. Save any command that has been referenced in that many events. During start-up, the shell sources `~/.history` into the history list, enabling history to be saved across log-ins. If the value of `savehist` is too large, the shell will be slow during start-up.

shell

Represent the file in which the shell resides. This is used in forking shells to interpret files that have execute bits set, but are not executable by the system. This variable is initialized to the (system-dependent) home of the shell.

status

Give the status returned by the last command. If it terminated abnormally, add 0200 to the status. Built-in commands that fail return exit status 1. All other built-in commands set status 0.

time

Control automatic timing of commands, if a numeric value is supplied. If set, print the user, system, and real times for any command that takes more than this many CPU seconds. Also print a utilization percentage (the ratio of user plus system times to real time) when the command terminates.

verbose

Print the words of each command after history substitution. This variable is set by the `-v` command line option to `csh`.

NON-BUILT-IN COMMAND EXECUTION

When a command to be executed is found to be something other than a built-in command, `cs` attempts to execute it via `execve(2)`. Each word in the variable `path` names a directory from which the shell attempts to execute the command. If it is given neither a `-c` nor a `-t` option, the shell hashes the names in these directories into an internal table so that it will only try an `exec` in a directory if the command potentially resides there. This greatly speeds command location when a large number of directories are present in the search path. For each directory component of `path` that does not begin with a slash (/), the shell concatenates with the given command name to form a pathname of a file which it then attempts to execute. The shell also does this if the internal hash table mechanism has been turned off (via `unhash`), or a `-c` or `-t` command line option was specified in `cs`.

Commands in parentheses are always executed in a sub-shell. Thus, `(cd ; pwd) ; pwd` prints the *home* directory, leaving you where you were (printing this after the *home* directory). On the other hand, `cd ; pwd` leaves you in the *home* directory. Commands in parentheses are most often used to prevent `chdir` from affecting the current shell.

If a file has execute permissions but is not an executable binary to the system, `cs` assumes it to be a file containing shell commands reads it (spawning a new shell to do so if `inprocess` is `unset`).

If there is an alias for `shell`, then the words of the alias will be prepended to the argument list to form the shell command. The first word of the alias should be the full pathname of the shell (e.g., `$Shell`). Note that this is a special, late-occurring, case of alias substitution, and it only allows words to be prepended to the argument list without modification.

COMMAND LINE OPTIONS

- `-c` Read commands from the (single) following argument that must be present. Place any remaining arguments in `argv`.
- `-e` Exit if any invoked command terminates abnormally or yields a non-zero exit status.
- `-f` Start up the shell more quickly than normal. Do not take time to search for or execute commands from the `.cshrc` file in the invoker's home directory.
- `-i` Make the shell interactive, prompting for its top-level input (even if it does not appear to be a terminal). shells are interactive without this option if their inputs and outputs are terminals.
- `-j` Run commands in-process.
- `-n` Parse commands, but do not execute them. This aids in syntactic checking of shell scripts.

- s Take command input from the standard input.
- t Read and execute a single line of input. A backslash (\) may be used to escape the newline at the end of this line and continue onto another line.
- v Set the verbose variable, causing command input to be echoed after history substitution.
- x Set the echo variable, so that commands are echoed immediately before execution.
- V Set the verbose variable even before *.cshrc* is executed.
- X Set the echo variable even before *.cshrc* is executed.

If argument 0 to the shell is a dash (-), then this is a log-in shell. If arguments remain after command line options are processed, but none of the -c, -i, -s, or -t options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by \$0. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, csh executes a standard shell if the first character of a script is not a pound sign (#), i.e., the script does not start with a comment. Remaining arguments initialize the argv variable.

SIGNAL HANDLING

Csh normally ignores quit signals. Jobs running detached, either by an ampersand (&) or the bg or %... & commands, are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shell's handling of interrupts and terminate signals in shell scripts can be controlled by onintr. Log-in shells catch the TERM (terminate) signal. Otherwise, this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a log-in shell is reading the file *.logout*.

CAUTIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to one-sixth the number of characters allowed in an argument list.

Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of alias substitutions on a single line to twenty.

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory. This can be misleading, since the job may have changed directories internally.

Shell built-in functions cannot be stopped and then restarted. Csh does not handle command sequences such as *a ; b ; c* gracefully when stopping is attempted. If you suspend *b*, the shell immediately executes *c*. This is especially noticeable if this expansion results from an alias. Place the sequence of commands in parentheses to force it to a sub-shell, i.e., (*a ; b ; c*).

Control over TTY output after processes are started is primitive.

Commands within loops, prompted for by a question mark (?), are not placed in the history list.

Control structure should be parsed rather than being recognized as built-in commands. This allows control commands to be placed anywhere, to be combined with the pipe character (|), and to be used with ampersand (&) and semicolon (;) metasyntax.

You cannot use the colon (:) modifiers on the output of command substitutions. More than one colon (:) modifier is not allowed on dollar sign (\$) substitutions.

Symbolic links fool the shell. In particular, pathnames that contain "." preceeded by a symbolic link always refer to the "real" parent directory and not the parent directory of the symbolic link.

FILES

<i>~/.cshrc</i>	<i>read at beginning of execution by</i>
<i>~/.login</i>	<i>read by log-in shell, after .cshrc</i>
<i>~/.logout</i>	<i>read by log-in shell, at log-out</i>
<i>/bin/sh</i>	<i>standard shell; for shell scripts not</i>
<i>/tmp/sh*</i>	<i>temporary file for <<</i>
<i>/etc/passwd</i>	<i>source of home directories for ~ name</i>

RELATED INFORMATION

sh(1), environ(5).

NAME

csplit – context split

USAGE

csplit [-s] [-k] [-f *prefix*] *file* *arg1* [... *argn*]

DESCRIPTION

Csplit reads a specified *file* and separates it into $n+1$ sections, defined by the arguments *arg1* ... *argn*. By default, it places the sections in *xx00* ... *xxn* (n may not be greater than 99). These sections comprise the following pieces of *file*:

- 00: From the start of *file* up to (but not including) the line referenced by *arg1*.
- 01: From the line referenced by *arg1* up to the line referenced by *arg2*.
- .
- .
- .
- $n+1$: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a – then standard input is used.

OPTIONS

- s Suppress the character counts for each file created.
- k Leave previously-created files intact (even if an error occurs when using **csplit**).
- f *prefix* Name the created files *prefix 00* ... *prefixn*. The default is *xx00* ... *xxn*.

ARGUMENTS

- /regexp/* Create a file for the section, from the current line, up to (but not including) the line containing the regular expression *regexp*. The current line becomes the line containing *regexp*. The argument may be followed by an optional + or – some number of lines (e.g., */Page/-5*).
- %regexp%* Same as */regexp/*, except that no file is created for the section.
- lnno* Create a file from the current line, up to (but not including) *lnno*. The current line becomes *lnno*.
- {*num*} Repeat argument. May follow any of the above arguments. If it follows a *regexp*-type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

EXAMPLES

To create four files, *cobol00* . . . *cobol03*, type the following:

```
# csplit -f cobol file '/procedure division/' /par5./ /par16./
```

To recombine the “split” files after they have been edited, use this:

```
# cat cobol0[0-3] > file
```

To overwrite the original file, type the following command:

```
# csplit -k file 100 {99}
```

To split the file at every 100 lines, up to 10,000 lines -- and to retain the created files if there are less than 10,000 lines, use this command:

```
# csplit -k prog.c '%main(%' '/'}/+1' {20}\NR
```

Assuming that *prog.c* follows the normal C coding convention of ending routines with a right brace (}) at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in *prog.c*.

CAUTIONS

Place all *rexp*-type arguments containing blanks or other characters meaningful to *sh* in appropriate quotes.

Regular expressions may not contain embedded newlines.

Csplit does not affect the original file; therefore, the file is not automatically removed after using csplit.

DIAGNOSTICS

Self-explanatory except for “arg – out of range”, which means that the given argument does not reference a line between the current position and the end of the file.

RELATED INFORMATION

ed(1)

sh(1)

regexp(5)

NAME

ctrace – C program debugger

USAGE

ctrace [*options*] [*file*]

DESCRIPTION

The ctrace command lets you follow the execution of a C program, statement by statement. The effect is similar to executing a Shell procedure with the `-x` option. It reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of ctrace in a temporary file because the `cc(1)` command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes, it is listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, and then by any output from the statement. When a loop is detected in the trace output, tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output.

OPTIONS

The most commonly-used options to ctrace are as follows:

- `-f functions` Trace only the specified *functions*.
- `-v functions` Trace all but the specified *functions*.

In using the `-v` option, you may add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. String arguments to the `string(3C)` functions and return values from `fgets(3S)`, `gets(3S)`, and `sprintf(3S)` are printed as strings. You can request that variables be printed in additional formats, if appropriate, by adding the following characters to the `-v` option:

- `o` (octal)
- `x` (hexadecimal)
- `u` (unsigned)
- `e` (floating point)

These options are used only in special circumstances:

- l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
- s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.
- t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). (See "Diagnostics" below.)
- P Run the C preprocessor on the input before tracing it. You can also use the -D, -I, and -U options to the cc(1) preprocessor.

Use the following *options* to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

- b Use only basic functions in the trace code, that is, those in ctype(3C), printf(3S), and string(3C). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have signal(2), fflush(3S), longjmp(3C), or setjmp(3C).
- p '*string*' Change the trace print function from the default of 'printf(' to *string*. For example, 'fprintf(stderr' sends the trace to the standard error output.
- r *f* Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the -p option).

CAUTIONS

You will get a ctrace syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

The ctrace program assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables (e.g., int EOF;) causes a syntax error.

Pointer values are always treated as pointers to character strings.

The `ctrace` program does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. When printing the value of an aggregate, `ctrace` may choose to print the address of an aggregate or use the wrong format (e.g., `%e` for a structure with two integer members).

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

EXAMPLES

If the file `lc.c` contains this C program:

```
1 #include <stdio.h>
2 main()      /* count lines in input */
3 {
4     int c, nl;
5
6     nl = 0;
7     while ((c = getchar()) != EOF)
8         if (c == '\n')
9             ++nl;
10    printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
#cc lc.c
#a.out
#1
(CTRL/D)
```

the program will be compiled and executed. (Note: CTRL/D, used above, applies as an end-of-file control key sequence only if your keyboard is mapped to `/sys/dm/sys5_keys`; otherwise, you must use a CTRL/Z instead.) The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke `ctrace` with these commands:

```
#ctrace lc.c >temp.c
#cc temp.c
#a.out
```

the output will be:

```

2 main()
6     nl = 0;
    /* nl == 0 */
7     while ((c = getchar()) != EOF)

```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```

    /* c == 49 or '1' */
8         if (c == '\n')
            /* c == 10 or '\n' */
9             ++nl;
            /* nl == 1 */
7     while ((c = getchar()) != EOF)
    /* c == 10 or '\n' */
8         if (c == '\n')
            /* c == 10 or '\n' */
9             ++nl;
            /* nl == 2 */
7     while ((c = getchar()) != EOF)

```

If you now enter an end-of-file character (CTRL-D) the final output will be:

```

    /* c == -1 */
10    printf("%d\n", nl);
    /* nl == 2 */
    return

```

Note that the program output printed at the end of the trace line for the *nl* variable. Also note the *return* comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

Since, by default, *ctrace* traces the entire program file, you do not normally have statement-by-statement control of the tracing, nor can you turn the tracing off and on when executing the traced program. You can do both of these by adding *ctroff()* and *ctron()* function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the CTRACE preprocessor variable.

For example:

```
#ifdef CTRACE
    if (c == '!' && i > 1000)
        ctron();
#endif
```

DIAGNOSTICS

Although this section contains diagnostic messages from `ctrace`, you should know that the traced code may also get an occasional warning message (or, in rare instances, an error message) from `cc(1)`. See `cc` for an explanation of appropriate compiler diagnostics.

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to prevent the C compiler “out of tree space; simplify expression” error. Use the `-t` option to increase this number.

warning: statement too long to trace

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use -P option

This is usually caused by `#ifdef/#endif` preprocessor statements in the middle of a C statement, or by a semicolon at the end of a `#define` preprocessor statement.

'if ... else if' sequence too long

Split the sequence by removing an `else` statement from the middle.

possible syntax error, try -P option

Use the `-P` option to preprocess the `ctrace` input, along with any appropriate `-D`, `-I`, and `-U` preprocessor options. (See “CAUTIONS” for more information.)

FILES

`/usr/lib/ctrace/runtime.c` run-time trace package

RELATED INFORMATION

<code>cc(1)</code>	<code>longjmp(3C)</code>
<code>signal(2)</code>	<code>printf(3S)</code>
<code>ctype(3C)</code>	<code>setjmp(3C)</code>
<code>fflush(3S)</code>	<code>string(3C)</code>

NAME

cu – call another UNIX system

USAGE

cu [*-sspeed*] [*-lline*] [*-h*] [*-t*] [*-d*] [*-m*] [*-o*] [*-e*] [*-n*] *telno* | *systemname* | *dir*

DESCRIPTION

Cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible ASCII file transfers.

After making the connection, **cu** runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with a tilde (~), passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with a tilde (~), passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with a tilde (~) have special meanings. Both the *transmit* and the *receive* processes are described in the sections below.

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using a double tilde (~~). For example, **uname(1)** can be executed on Z, X, and Y as follows:

```
uname
Z
~!uname
X
~~!uname
Y
```

In general, a tilde causes the command to be executed on the original machine; a double tilde causes the command to be executed on the next machine in the chain.

The DOMAIN/IX version of **cu** supports the Vadic 212 Autodialer.

OPTIONS

- sspeed** Specify the transmission speed(110, 150, 300, 600, 1200, 4800, or 9600); 300 is the default value. Most modems are either 300 or 1200 baud. Directly-connected lines may be set to a speed higher than 1200 baud.
- lline** Specify a device name to use as the communication line. Can be used to override searching for the first available line having the right speed. When this option is used without the **-s** option, the speed of a line is taken from the */usr/lib/uucp/L-devices* file. With the **-s** option, the *L-devices* file is searched for the requested speed for the requested line. If possible, the connection is made at the requested speed; otherwise, an

error message is printed and the call is not made. The specified device is generally a directly-connected asynchronous line; therefore, a phone number is not required but the string *dir* may be used to specify a null *acu*. If the specified device is associated with an auto dialer, a phone number must be provided.

- h Emulate local echo, supporting calls to other computer systems that expect terminals to be set to half-duplex mode.
- t Set appropriate mapping of carriage-return to carriage-return-line-feed pairs. Used when dialing an ASCII terminal set to auto answer.
- d Print diagnostic traces.
- e Generate even parity for data sent to the remote.
- o Generate odd parity for data sent to the remote.
- m Designate a direct line with modem control.
- n Request the phone number to be dialed from the user rather than taking it from the command line.

SPECIAL ARGUMENTS

- telno* When using an automatic dialer, *telno* is the telephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.
- systemname* *Systemname* is a **uucp** system name that may be used rather than a phone number. **Cu** obtains an appropriate direct line or phone number from */usr/lib/uucp/L.sys* (the appropriate baud rate is also read along with phone numbers). **Cu** tries each phone number or direct line for *systemname* in the *L.sys* file until a connection is made or all the entries are tried.
- dir* Using *dir* ensures that **cu** uses the line specified by the **-l** option.

TRANSMIT PROCESS

The *transmit* process interprets the following:

- ~. Terminate the conversation.
- ~! Escape to the shell on the local system.
- ~!cmd... Run *cmd* on the local system, via the **-c** option to the **sh(1)** command.
- ~\$cmd... Run *cmd* locally and send its output to the remote system.

- ~ %cd Change the directory on the local system. **NOTE:** A ~!cd causes the command to be run by a sub-Shell, which was probably not what was intended.
- ~ %take *from* [*to*] Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
- ~ %put *from* [*to*] Copy file *from* (on the local system) to file *to* on the remote system. If *to* is omitted, the *from* argument is used in both places.
- ~~ ... Send the line ~... to the remote system.
- ~ %break Transmit a *break* to the remote system.
- ~ %nostop Toggle between DC3/DC1 input control protocol and no input control. This is useful when the remote system is one that does not respond properly to the DC3 and DC1 characters.

RECEIVE PROCESS

The *receive* process normally copies data from the remote system to its standard output. A line from the remote beginning with ~> initiates an output diversion to a file. The complete sequence is as follows:

```
~>[>]: file
zero or more lines to be written to file
~>
```

Data from the remote is diverted (or appended, if >> is used) to *file*. The trailing ~> terminates the diversion.

Using ~ %put requires stty(1) and cat(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places. There is an artificial slowing of transmission during the ~ %put operation, so that loss of data is unlikely.

Using ~ %take requires echo(1) and cat(1) on the remote system. The stty tabs mode should also be set on the remote system, if tabs are to be copied without expansion.

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud (default speed is 300), use the following command:

```
# cu -s1200 9=2015551212
```


To log in to a system connected by a direct line, type this (where *XX* is a valid TTY number):

```
# cu -l /dev/ttyXX dir
```

To dial a system with the specific line and a specific speed, type this (where *XX* is a valid TTY number):

```
# cu -s1200 -l /dev/ttyXX dir
```

To dial a system using a specific line, execute the following command (where *XX* is a line number):

```
# cu -l /dev/culXX 2015551212
```

To use a system name, use this command (where *YYYYZZZ* is the name of the system):

```
# cu YYYYZZZ
```

CAUTIONS

If you *cu* to a DOMAIN node whose default start-up shell is */com/sh* (as opposed to */bin/sh* or */bin/csh*), you should either: 1) change your command search rules (i.e., do a

```
csr -a /bin /usr/bin ...
```

inside the AEGIS Shell) so that the *cu transmit* process can properly locate DOMAIN/IX commands, or 2) have the remote start-up AEGIS Shell invoke a DOMAIN/IX Shell (i.e., */bin/sh*) so that the *cu receive* process can properly parse the request (since the tilde character has a special meaning in the AEGIS Shell).

Cu buffers input internally.

FILES

```
/usr/lib/uucp/L.sys  
/usr/lib/uucp/L-devices  
/usr/spool/uucp/LCK.. (tty-device)  
/dev/null
```

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

RELATED INFORMATION

cat(1), *echo*(1), *stty*(1), *uname*(1), *uucp*(1C).

NAME

cut – cut out selected fields from each line of a file

USAGE

cut **-c** *list* [*file1 file2 ...*]

cut **-f** *list* [**-d** *char*] [**-s**] [*file1 file2 ...*]

DESCRIPTION

Cut extracts columns from a table or fields from each line of a file. The length of the fields specified by *list* can be fixed (i.e., character positions as on a punched card), or varied from line-to-line. They may also be marked with a field delimiter character such as *tab*. You may use **cut** as a filter. If you specify no files, it reads from the standard input.

Cut is useful, but has its limitations. To make horizontal “cuts” through a file and then put the pieces back together, you should use the **grep(1)** and **paste(1)** commands respectively. You can reorder columns in a table by using **cut** and **paste**.

OPTIONS

- | | |
|-----------------------|--|
| <i>list</i> | Specify a comma-separated list of integer field numbers in increasing order, with an optional dash (–) to indicate ranges (e.g., <i>1,4,7</i> ; <i>1–3,8</i> ; <i>–5,10</i> (short for <i>1–5,10</i>); or <i>3–</i> (short for third through last field). |
| -c <i>list</i> | Specify a <i>list</i> of character positions (e.g., -c1–72 passes the first 72 characters of each line). |
| -f <i>list</i> | Specify a list of fields separated in the file by a delimiter character (see the -d option); e.g., -f1,7 copies the first and seventh field only. Lines with no field delimiters are passed through intact (useful for table sub-headings), unless -s is specified. |
| -d <i>char</i> | Specify the field delimiter equal to <i>character</i> (-f option only). The default is <i>tab</i> . A space or other characters with special meaning to the Shell must be quoted. |
| -s | Suppress lines with no delimiter characters in case of the -f option. Unless specified, lines with no delimiters are passed through untouched. |

EXAMPLES

To map user IDs to names, type the following:

cut -d: -f1,5 /etc/passwd

To set *name* to your current log-in name (this is run in the Shell), type the following:

```
# name='who am i | cut'
```

CAUTIONS

You must always specify either the `-c` or `-f` option on a `cut` command line.

DIAGNOSTICS

“line too long”

The line contains more than 1023 characters or fields.

“bad list for c/f option”

Missing a `-c` or `-f` option or a correctly-specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

“no fields”

The *list* is empty.

RELATED INFORMATION

`grep(1)`, `paste(1)`.

NAME

cw, **checkcw** – prepare constant-width text for **troff**

USAGE

cw [-lxx] [-rxx] [-fn] [-tl+t] [-d] *file(s)*
checkcw [-lxx] [-rxx] *file(s)*

DESCRIPTION

Cw is a preprocessor for **troff**(1) input files containing text to be typeset in the constant-width (CW) font.

Since text typeset using the CW font resembles the output of terminals and line printers, this font is used to typeset program examples and computer output in user manuals, programming texts, etc. It has been designed to look quite distinctive when used with the Times Roman font.

Because the CW font contains a “non-standard” set of characters, and text typeset with it requires different character and interword spacing than is used for “standard” fonts, documents that use the CW font must be preprocessed by **cw**.

The CW font contains the following 94 printable ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&'()*+@.,/:;=?[]_~"<>{}#
```

plus eight non-ASCII characters represented by four-character **troff**(1) names (in some cases attaching these names to “non-standard” graphics), as follows:

<i>Character</i>	<i>Symbol</i>	<i>Troff Name</i>
“Cents” sign	¢	(ct
EBCDIC “not” sign	¬	(no
Left arrow	←	(<-
Right arrow	→	(->
Down arrow	↓	(da
Vertical single quote	'	(fm
Control-shift indicator	†	(dg
Visible space indicator	□	(sq
Hyphen	-	(hy

The hyphen is a synonym for a plain minus sign (-). Certain versions of `cw` recognize two additional names: `\ua` for an up arrow and `\lh` for a diagonal left-up (home) arrow.

`Cw` recognizes five request lines, as well as user-defined delimiters. The request lines look like `troff(1)` macro requests, and are copied in their entirety by `cw` onto its output. Thus, you can define them as `troff(1)` macros. In fact, the `.CW` and `.CN` macros *should* be defined this way.

`Cw` reads the standard input when no *files* are specified, so it can be used as a filter. A typical use is as follows:

```
cw files | troff ...
```

`Checkcw` checks that left and right delimiters, as well as the `.CW/.CN` pairs, are properly balanced. It prints out all offending lines.

This version of the commands is an older one (pre-System V), because the System V version is only available through the AT&T Documenter's Workbench for which Apollo has not yet been licensed.

REQUESTS

- .CW** Set start of text in the CW font. Use `.CW` to cause a break. Take precisely the same options, in precisely the same format, that are available on the `cw` command line.
- .CN** Set end of text in the CW font. Use `.CN` to cause a break. Take the same options that are available on the `cw` command line.
- .CD** Change delimiters and/or settings of other options. Take the same options that are available on the `cw` command line.
- .CParg1 arg2 arg3 ...argn**
Concatenate all arguments that are delimited like `troff(1)` macro arguments. Set the odd-numbered arguments in the CW font, and the even-numbered ones in the prevailing font.
- .PCarg1 arg2 arg3 ...argn**
Perform the same function as `.CP`, except set the even-numbered (rather than odd-numbered) arguments in the CW font.

The `.CW` and `.CN` requests are meant to bracket text (e.g., a program fragment) to be typeset in the CW font "as is." Normally, `cw` operates in the *transparent* mode. In this mode, except for the `.CD` request and the nine special four-character names listed in the table above, every character between `.CW` and `.CN` request lines stands for itself.

In particular, cw arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) and ligatures (fi, ff, etc.) everywhere to be "hidden" from troff(1). The transparent mode can be turned off (see below), in which case normal troff(1) rules apply. In any case, the effect of the font changes generated by the .CW and .CN requests are invisible to you.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

DELIMITERS

You can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text. Cw treats text enclosed by delimiters just like text bracketed by .CW/.CN pairs. For aesthetic reasons, however, spaces in text bracketed by .CW/.CN pairs have the same width as any other CW character. Spaces between delimiters are half as wide, so that they have the same width as spaces in the prevailing text (but they are *not* adjustable).

Delimiters have no special meaning inside .CW/.CN pairs.

OPTIONS

- lxx Specify the one- or two-character string *xx* as the left delimiter. If *xx* is omitted, specify the left delimiter as undefined, which it is initially.
- rxx Specify the one- or two-character string *xx* as the right delimiter. The left and right delimiters may (but need not) be different.
- fn Mount the CW font in font position *n*. Acceptable values for *n* are 1, 2, and 3 (the default is 3, replacing the bold font). This option is only useful at the beginning of a document.
- t Turn transparent mode *off*.
- +t Turn transparent mode *on* (initial default).
- d Print current option settings on file descriptor 2 in the form of troff(1) comment lines. This option is meant for debugging.

CAUTIONS

Text preprocessed by cw requires that it be set typeset on a machine equipped with the CW font.

It is not wise to use periods (.) or backslashes (\) as delimiters.

Some CW characters do not concatenate gracefully with certain Times Roman characters; e.g., a CW ampersand (&) followed by a Times Roman comma(.). In such cases, troff(1) half- and quarter-spaces must be carefully used.

Cw output is often difficult to read.

FILES

/usr/lib/font/ftCW

CW font-width table

RELATED INFORMATION

eqn(1)

mm(1)

tbl(1)

troff(1)

NAME

cxref – generate a C program cross-reference

USAGE

cxref [*options*] *files*

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. It uses a special version of `cpp(1)` to include `#define`'d information in its symbol table. Cxref then prints, on standard output, a separate or combined listing of all symbols (auto, static, and global) in each file. An asterisk (*) precedes the declaring reference for each symbol.

OPTIONS

- | | |
|------------------------------------|--|
| -c | Print a combined cross-reference of all input files. |
| -w<num> | Format output no wider than <num> (decimal) columns. Default to 80 if <num> is not specified or is less than 51. |
| -ofile | Direct output to named <i>file</i> . |
| -s | Operate silently; do not print input filenames. |
| -t | Produce format listing for 80-column width. |
| -Uname | Remove any initial definition of <i>name</i> , where <i>name</i> is a reserved symbol predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system: ibm, gcos, os, tss, unix
hardware: apollo, interdata, pdp11, u370, u3b, u3b5, vax
UNIX system variant: RES, RT
lint(1): lint |
| -Dname
-Dname=def | Define <i>name</i> as if by a <code>#define</code> directive. If no <code>=def</code> is given, define <i>name</i> as 1. This option has lower precedence than the -U option. That is, if the same name is used in both a -U option and a -D option, the name is undefined regardless of the order of the options. |
| -Idir | Change the algorithm for searching for <code>#include</code> files whose names do not begin with / to look in <i>dir</i> before looking in the directories on the standard list. Thus, <code>#include</code> files whose names are enclosed in double quotes (" ") are first sought in the directory of the file with the <code>#include</code> line, then in directories named in -I options, and finally in directories on a standard list. |

For **#include** files whose names are enclosed in **<>**, the directory of the file with the **#include** line is not searched.

CAUTIONS

Cxref considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program which **#includes** *ctype.h*, contains many declarations of the variable *c*.

FILES

/usr/lib/xcpp special version of the C preprocessor

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile certain files.

RELATED INFORMATION

cc(1), cpp(1).

NAME

date – print the date

USAGE

date [*+format*]

DESCRIPTION

Date prints the current date and time. If you specify an argument beginning with a plus (+), the output of **date** is user-controlled. The output format is similar to that of the first argument to **printf(3S)**. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and is replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a newline character.

FIELD DESCRIPTORS

n	Newline character
t	Tab character
m	Month of year – 01 to 12
d	Day of month – 01 to 31
y	Last 2 digits of year – 00 to 99
D	Date as mm/dd/yy
H	Hour – 00 to 23
M	Minute – 00 to 59
S	Second – 00 to 59
T	Time as HH:MM:SS
j	Day of year – 001 to 366
w	Day of week – Sunday = 0
a	Abbreviated weekday – Sun to Sat
h	Abbreviated month – Jan to Dec
r	Time in AM/PM notation

CAUTIONS

The DOMAIN/IX version of `date` does not allow you to set the system date and time interactively. Your node must be shut down and the DOMAIN `calendar` stand-alone utility run in order to make the change. Refer to the DOMAIN System Command Reference for information on this.

RELATED INFORMATION

`printf(3S)`.

NAME

dbx – debugger

SYNOPSIS

dbx [-r] [-i] [-I *dir*] [-no_src] [-no_frame] [-c *file*] [*objfile*]

DESCRIPTION

Dbx is a tool for source level debugging and execution of programs under DOMAIN/IX. The *objfile* is an object file produced by a compiler with the appropriate flag (usually "-g") specified to produce symbol information in the object file. The machine level facilities of dbx can be used on any program.

The object file contains a symbol table that includes the name of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If the file ".dbxinit" exists in the current directory then the debugger commands in it are executed. Dbx also checks for a ".dbxinit" in the user's home directory if there isn't one in the current directory.

Dbx will create a separate transcript pad for debugger interactions unless the -no_frame option is specified. Dbx will also create a window to display source code unless the -no_src is specified.

OPTIONS

The command line options and their meanings are:

- r Execute *objfile* immediately. If it terminates successfully dbx exits. Otherwise the reason for termination will be reported and the user offered the option of entering the debugger or letting the program fault. Dbx will read from "/dev/tty" when -r is specified and standard input is not a terminal.
- i Force dbx to act as though standard input is a terminal.
- I *dir* Add *dir* to the list of directories that are searched when looking for a source file. Normally dbx looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the use command.
- c *file* Execute the dbx commands in the *file* before reading from standard input.
- no_src Disable source display.
- no_frame Do not create a separate debugger transcript pad.

Unless **-r** is specified, **dbx** just prompts and waits for a command.

Execution and Tracing Commands

run [*args*] [< *filename*] [> *filename*]

rerun [*args*] [< *filename*] [> *filename*]

Start executing *objfile*, passing *args* as command line arguments; < or > can be used to redirect input or output in the usual manner. When **rerun** is used without any arguments the previous argument list is passed to the program; otherwise it is identical to **run**. If *objfile* has been written since the last time the symbolic information was read in, **dbx** will read in the new information.

trace [*in procedure/function*] [*if condition*]

trace *source-line-number* [*if condition*]

trace *procedure/function* [*in procedure/function*] [*if condition*]

trace *expression* **at** *source-line-number* [*if condition*]

trace *variable* [*in procedure/function*] [*if condition*]

Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. Source line numbers in a file other than the current one must be preceded by the name of the file in quotes and a colon, e.g. "mumble.p":17.

If the argument is a procedure or function name then every time it is called, information is printed telling what routine called it, from what source line it was called, and what parameters were passed to it. In addition, its return is noted, and if it's a function then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause "*in procedure/function*" restricts tracing information to be printed only while executing inside the given procedure or function.

Condition is a boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

stop if *condition*

stop at *source-line-number* [*if condition*]

stop in *procedure/function* [*if condition*]

stop *variable* [*if condition*]

Stop execution when the given line is reached, procedure or function called, variable changed, or condition true.

status [*> filename*]

Print out the currently active **trace** and **stop** commands.

delete *command-number ...*

The traces or stops corresponding to the given numbers are removed. The numbers associated with traces and stops are printed by the **status** command.

catch *number*

catch *signal-name*

ignore *number*

ignore *signal-name*

Start or stop trapping a signal before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. A signal may be specified by number or by a name (e.g., SIGINT). Signal names are case insensitive and the "SIG" prefix is optional. By default all signals are trapped except SIGCONT, SIGCHILD, SIGALRM and SIGKILL.

cont *integer*

cont *signal-name*

Continue execution from where it stopped. If a signal is specified, the process continues as though it received the signal. Otherwise, the process is continued as though it had not been stopped. Execution cannot be continued if the process has "finished", that is, called the standard procedure "exit".

step Execute one source line.

next Execute up to the next source line. The difference between this and **step** is that if the line contains a call to a procedure or function the **step** command will stop at the beginning of that block, while the **next** command will not.

return [*procedure*]

Continue until a return to *procedure* is executed, or until the current procedure returns if none is specified.

call *procedure(parameters)*

Execute the object code associated with the named procedure or function.

Printing Variables and Expressions

Names are resolved first using the static scope of the current function, then using the dynamic scope if the name is not defined in the static scope. If static and dynamic searches do not yield a result, an arbitrary symbol is chosen and the message "[using *qualified name*]" is printed. The name resolution procedure may be overridden by qualifying an identifier with a block name, e.g., "*module.variable*". For C, source files are treated as modules named by the file name in uppercase with the string "_c" substituted for ".c", eg., "x.c" => "X_C".

Expressions are specified with an approximately common subset of C and Pascal (or equivalently Modula-2) syntax. Indirection can be denoted using either a prefix "*" or a postfix "^" and array expressions are subscripted by brackets ("[]"). The field reference operator (".") can be used with pointers as well as records, making the C operator "->" unnecessary (although it is supported).

Types of expressions are checked; the type of an expression may be overridden by using "*type-name(expression)*". When there is no corresponding named type the special constructs "&*type-name*" and "\$\$*tag-name*" can be used to represent a pointer to a named type or C structure tag.

assign *variable* = *expression*

Assign the value of the expression to the variable.

dump [*procedure*] [> *filename*]

Print the names and values of variables in the given procedure, or the current one if none is specified. If the procedure given is ".", then the all active variables are dumped.

print *expression* [, *expression* ...]

Print out the values of the expressions.

whatis *name*

Print the declaration of the given name, which may be qualified with block names as above.

which *identifier*

Print the full qualification of the given identifier, i.e. the outer blocks that the identifier is associated with.

up [*count*]

down [*count*]

Move the current function, which is used for resolving names, up or down the stack *count* levels. The default *count* is 1.

where Print out a list of the active procedures and function.

whereis *identifier*

Print the full qualification of all the symbols whose name matches the given identifier. The order in which the symbols are printed is not meaningful.

Accessing Source Files***/regular expression[/]******?regular expression[?]***

Search forward or backward in the current source file for the given pattern.

edit [*filename*]**edit *procedure/function-name***

Invoke an editor on *filename* or the current source file if none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable EDITOR to the name of the desired editor.

file [*filename*]

Change the current source file name to *filename*. If none is specified then the current source file name is printed.

func [*procedure/function*]

Change the current function. If none is specified then print the current function. Changing the current function implicitly changes the current source file to the one that contains the function; it also changes the current scope used for name resolution.

list [*source-line-number* [, *source-line-number*]]**list *procedure/function***

List the lines in the current source file from the first line number to the second inclusive. If no lines are specified, the next 10 lines are listed. If the name of a procedure or function is given lines *n-k* to *n+k* are listed where *n* is the first statement in the procedure or function and *k* is small.

use *directory-list*

Set the list of directories to be searched when looking for source files. The *directory-list* is used if the specified file cannot be found, or if the file is found but the modified time does not match the time in the object module. If a file is found using *directory-list*, or if the file's modified time is different then the source display banner will display the filename being displayed as well as the stored filename in parentheses.

Command Aliases and Variables

alias *name name*

alias *name "string"*

alias *name (parameters) "string"*

When commands are processed, **dbx** first checks to see if the word is an alias for either a command or a string. If it is an alias, then **dbx** treats the input as though the corresponding string (with values substituted for any parameters) had been entered. For example, to define an alias "rr" for the command "rerun", one can say

```
alias rr rerun
```

To define an alias called "b" that sets a stop at a particular line one can say

```
alias b(x) "stop at x"
```

Subsequently, the command "b(12)" will expand to "stop at 12".

set *name [= expression]*

The **set** command defines values for debugger variables. The names of these variables cannot conflict with names in the program being debugged, and are expanded to the corresponding expression within other commands. The following variables have a special meaning:

\$hexchars

\$hexints

\$hexoffsets

\$hexstrings

When **set**, **dbx** prints out characters, integers, offsets from registers, or character pointers respectively in hexadecimal.

\$listwindow

The value of this variable specifies the number of lines to list around a function or when the **list** command is given without any parameters. This value is also used when displaying source in the source window. The current line is positioned so that as much of the **listwindow** as possible is visible. Its default value is 10.

\$unsafecall

\$unsafeassign

When "**\$unsafecall**" is set, strict type checking is turned off for arguments to subroutine or function calls (*e.g.* in the **call** statement). When "**\$unsafeassign**"

is set, strict type checking between the two sides of an `assign` statement is turned off. These variables should be used only with great care, because they severely limit `dbx`'s usefulness for detecting errors.

unalias *name*

Remove the alias with the given name.

unset *name*

Delete the debugger variable associated with *name*.

Machine Level Commands

tracei [*address*] [*if cond*]

tracei [*variable*] [*at address*] [*if cond*]

stopi [*if cond*]

stop at address [*if cond*]

Turn on tracing or set a stop using a machine instruction address.

stepi

nexti Single step as in `step` or `next`, but do a single instruction rather than source line.

address ,address/ [mode]

address / [count] [mode]

Print the contents of memory starting at the first *address* and continuing up to the second *address* or until *count* items are printed. If the address is ".", the address following the one printed most recently is used. The *mode* specifies how memory is to be printed; if it is omitted the previous mode specified is used. The initial mode is "X". The following modes are supported:

i	print the machine instruction
d	print a short word in decimal
D	print a long word in decimal
o	print a short word in octal
O	print a long word in octal
x	print a short word in hexadecimal
X	print a long word in hexadecimal
b	print a byte in octal
c	print a byte as a character
s	print a string of characters terminated by a null byte
f	print a single precision real number
g	print a double precision real number

Symbolic addresses are specified by preceding the name with an "&". Registers are denoted by \$D0-\$D7, for the data registers, and \$A0-\$A7, for the address registers. For convenience, \$DB, \$SB, \$SP, and \$PC are also available. Addresses may be expressions made up of other addresses and the operators "+", "-", and indirection (unary "*").

Miscellaneous Commands

help Print out a synopsis of **dbx** commands.

quit Exit **dbx**.

sh *command-line*

Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

source *filename*

Read **dbx** commands from the given *filename*.

FILES

<i>a.out</i>	object file
<i>.dbxinit</i>	initial commands

RELATED INFORMATION

cc(1)

COMMENTS

Assignments to structures with bit fields does not work, and assigning through a pointer variable may cause **dbx** to have a stack underflow and abort.

Some problems remain with the support for individual languages. Fortran problems include: inability to assign to logical, logical*2, complex and double complex variables; inability to represent parameter constants which are not type integer or real; peculiar representation for the values of dummy procedures (the value shown for a dummy procedure is actually the first few bytes of the procedure text; to find the location of the procedure, use "&" to take the address of the variable).

NAME

dc – desk calculator

USAGE

dc [*file*]

DESCRIPTION

Dc is an arbitrary-precision arithmetic package. It normally operates on decimal integers, but you may specify an input base, output base, and a number of fractional digits to be maintained. Bc(1) is the preprocessor to this package, providing infix notation, a C-like syntax that implements functions, and reasonable control structures for programs. Dc is, fundamentally, a stacking calculator. If an argument is given, input is taken from that *file* until its end, then from the standard input.

OPERATORS

<i>number</i>	The value of the number pushed on the stack. A number is an unbroken string of the digits 0–9. May be preceded by an underscore (_) to signify a negative number. May contain decimal points.
+ - / * ^	Add (+), subtract (-), multiply (*), divide (/), remainder (), or exponentiate (^) the top two values on the stack. Pop the two entries off the stack; push the result on the stack in their place. Ignore any fractional part of an exponent.
sx	Pop the top of the stack and store it in a register named <i>x</i> , where <i>x</i> may be any character. If the <i>s</i> is capitalized, treat <i>x</i> as a stack and the value is pushed on it.
lx	Push the value in register <i>x</i> to the stack. The register <i>x</i> remains unaltered. All registers start with zero value. If the <i>l</i> is capitalized, register <i>x</i> is treated as a stack and its top value is popped onto the main stack.
d	Duplicate the top value on the stack.
p	Print the top value on the stack. Do not change the top value.
P	Interpret top of the stack as an ASCII string, remove the string, and then print it.
f	Print all values on the stack.
q	Exit the program. If executing a string, pop the recursion level by two.
Q	Pop the top value on the stack and the string execution level by that value.

x	Treat the top element of the stack as a character string, and execute it as a string of dc commands.
X	Replace the number on the top of the stack with its scale factor.
[...]	Put the bracketed ASCII string onto the top of the stack.
<x >x =x	Pop the top two elements of the stack and compare them. Evaluate register x if the elements obey the stated relation.
v	Replace the top element on the stack by its square root. Take any existing fractional part of the argument into account, but otherwise ignore the scale factor.
!	Interpret the rest of the line as a shell command.
c	Pop all values on the stack.
i	Pop the top value on the stack and use it as the number radix for further input. Push the input base on top of the stack.
o	Pop the top value on the stack and use it as the number radix for further output.
O	Push the output base on top of the stack.
k	Pop the top of the stack and use that value as a non-negative scale factor. Print the appropriate number of places output, and maintain them during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base is reasonable if all are changed together.
z	Push the stack level onto the stack.
Z	Replace the number on the top of the stack with its length.
?	Take a line of input from the input source (usually the terminal) and execute it.
;;	Used by bc for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

DIAGNOSTICS

"x is unimplemented"	x is an octal number.
"Stack empty"	Not enough elements on the stack to do function.
"Out of space"	Free list is exhausted (too many digits).
"Out of headers"	Too many numbers being retained.
"Out of pushdown"	Too many items on the stack.
"Nesting Depth"	Too many levels of nested execution.

RELATED INFORMATION

bc(1).

NAME

dd – convert and copy a file

USAGE

dd [*option=value*] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. By default, it uses the standard input and output. You may specify the input and output block size. After completion, **dd** reports the number of whole and partial input and output blocks.

OPTION/VALUE PAIRS

ibs=<i>n</i>	Input block size <i>n</i> bytes; 512 is the default.
obs=<i>n</i>	Output block size; 512 is the default.
bs=<i>n</i>	Set both input and output block size, superseding ibs and obs ; also, if no conversion is specified, it is particularly efficient since no in-core copy has to be done.
cbs=<i>n</i>	Conversion buffer size; used only if conv=ascii or conv=ebcdic is specified. In the former case, <i>cbs</i> characters are placed into the conversion buffer, converted to ASCII, and trimmed of any trailing blanks. Newlines are then added before sending the line to the output. In the latter case, ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks are added to make up an output block of size <i>cbs</i> .
skip=<i>n</i>	Skip <i>n</i> input blocks before starting copy.
seek=<i>n</i>	Seek <i>n</i> blocks from the beginning of the output file before copying.
count=<i>n</i>	Copy only <i>n</i> input blocks.
conv=ascii	Convert EBCDIC to ASCII.
ebcdic	Convert ASCII to EBCDIC.
ibm	Map ASCII to EBCDIC in a slightly different way than the above case.
lcase	Map alphabetics to lowercase.
ucase	Map alphabetics to uppercase.
swab	Swap every pair of bytes.
noerror	Do not stop processing on an error.
sync	Pad every input block to ibs .
... , ...	Represent several comma-separated conversions.

Where sizes are specified, a number of bytes is expected. A number may end with k, b, or w to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by x to indicate a product.

The ASCII/EBCDIC conversion tables are taken from the 256-character standard of the CACM (November, 1968). The *ibm* conversion, while less accepted as a standard, corresponds better to certain IBM print train conventions.

EXAMPLE

To read an EBCDIC tape blocked with ten 80-byte EBCDIC card images per block into the ASCII file *x*, use the following:

```
# dd if=/dev/rmt0 of=x ibs=800 cbs=80
```

CAUTIONS

DOMAIN/IX does not support some raw I/O devices typically used with *dd*.

Newlines are inserted only on conversion to ASCII. Padding is done only on conversion to EBCDIC. These should be separate options.

DIAGNOSTICS

f+p blocks in(out) numbers of full and partial blocks read(written)

RELATED INFORMATION

cp(1).

NAME

delta – make a delta (change) to an SCCS file

USAGE

delta /-r [*SID*] [-s] [-n] [-glist] [-m [*mrlist*]] [-y [*comment*]] [-p] *files*

DESCRIPTION

Delta permanently introduces into the named SCCS file changes made to the file retrieved by get(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, delta behaves as though each file in the directory were specified as a named file, except that it silently ignores non-SCCS and unreadable files. If you specify a dash (-) in place of a filename, it reads from the standard input, interpreting each line as the name of an SCCS file to be processed.

Delta issues prompts on the standard output, depending on the presence of certain options in the SCCS file. See admin(1) for more information.

OPTIONS

The options described below apply independently to each named file.

- r*SID* Uniquely identify which delta is to be made to the SCCS file. This option is necessary only if two or more outstanding get commands for editing (get -e) on the same SCCS file are done by the same person (log-in name). The *SID* value specified with this keyletter is either the *SID* specified on the get command line or the *SID* to be made as reported by the get(1) command. If the specified *SID* is ambiguous or if it was omitted when it was required to be present, a diagnostic results.
- s Do not print the created delta's *SID* or the number of lines inserted, deleted, and unchanged in the SCCS file on standard output.
- n Retain the edited *g-file* normally removed at completion of delta processing.
- glist Specify a *list* of deltas to be *ignored* when the file is accessed at the change level (*SID*) created by this delta. See get(1) for the definition of *list*.
- m [*mrlist*] If the SCCS file has the v flag set, then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta. See admin(1) for more information.

If -m is not used and the standard input is a terminal, the prompt "MRs?" is issued on the standard output before the standard input is

read; if the standard input is not a terminal, no prompt is issued. This particular prompt always precedes the prompt for comments (see the `-y` option). MRs in a list are separated by blanks and/or tab characters. An unescaped newline character terminates the MR list.

Note that if the `v` flag has a value, it is taken to be the name of a program (or Shell procedure) that validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, `delta` assumes that the MR numbers were not all valid, and it terminates.

`-y [comment]` Specify arbitrary text to describe the reason for making the delta. A null string is considered a valid *comment*.

If `-y` is not specified and the standard input is a terminal, the prompt `"comments?"` is issued on the standard output before the standard input is read. If the standard input is not a terminal, no prompt is issued. An unescaped newline character terminates the *comment* text.

`-p` On the standard output, print the SCCS file differences before and after the delta is applied. Use a `diff(1)` format.

CAUTIONS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS and will cause an error if it is not escaped. Refer to `sccsfile(4)` for more information.

Avoid a `get` of many SCCS files, followed by a `delta` of those files, when the `get` generates a large amount of data. Instead, multiple `get/delta` sequences should be used.

If you specify the standard input on the `delta` command line, you must use the `-m` and/or `-y` keyletters.

Comments are limited to text strings of 512 characters.

FILES

<i>g-file</i>	Existed before the execution of <code>delta</code> ; removed after completion of <code>delta</code> .
<i>p-file</i>	Existed before the execution of <code>delta</code> ; may exist after completion of <code>delta</code> .
<i>q-file</i>	Created during the execution of <code>delta</code> ; removed after completion of <code>delta</code> .
<i>x-file</i>	Created during the execution of <code>delta</code> ; renamed to SCCS file after completion of <code>delta</code> .
<i>z-file</i>	Created during the execution of <code>delta</code> ; removed during the execution of <code>delta</code> .

d-file Created during the execution of **delta**; removed after completion of **delta**.

/usr/bin/bdiff Program to compute differences between the “gotten” file and the *g-file*.

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

admin(1), **bdiff(1)**, **cdc(1)**, **get(1)**, **help(1)**, **prs(1)**, **rmDEL(1)**, **scsfile(4)**.

NAME

deroff – remove **nroff/troff**, **tbl**, and **eqn** constructs

USAGE

deroff [**-w**] [**-mm**] [**-ms**] [**-ml**] [*files*]

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all **troff**(1) requests, macro calls, backslash constructs, **eqn**(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and **tbl**(1) descriptions, and writes the remainder of the file on the standard output. **Deroff** follows chains of included files (**.so** and **.nx troff** commands). If a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, **deroff** reads the standard input.

DOMAIN/IX supports the UNIX System III version of **deroff**

OPTIONS

- | | |
|------------|--|
| -mm | Interpret the UNIX memorandum-style macros in mm (1) so that only running text (no macro lines) is output. |
| -ms | Interpret the UNIX manuscript-style (ms) macros so that only running text is output. |
| -ml | Perform the same function as the -mm option, but also delete lists associated with the mm macros. |
| -w | Output a word list, one “word” per line, with all other characters deleted. By default, output follows the original, with the deletions mentioned above. In text, a “word” is any string that contains at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('). In a macro call, however, a “word” is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from “words.” |

CAUTIONS

Deroff is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The **-ml** option does not handle nested lists correctly.

DEROFF (1)

DOMAIN/IX SYS5

DEROFF (1)

RELATED INFORMATION
eqn(1), tbl(1), troff(1).

NAME

diff – differential file comparator

USAGE

diff [-efbh] *file1 file2*

diff [-efbh] *dir1 dir2*

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If you specify a dash (-) in place of *file1* (*file2*), it reads from the standard input. If *file1* (*file2*) is a directory, diff searches a file in that directory with the name *file2* (*file1*).

The normal output of diff contains lines which look similar to this:

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

These lines resemble ed(1) commands to convert *file1* into *file2*. The a stands for add, the d for delete, and the c for change. The numbers after the letters a, b, and c pertain to *file2*. In fact, by exchanging a for d and reading backward one may ascertain equally how to convert *file2* into *file1*. Identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines are all the lines affected in the first file (flagged by <), then all the lines affected in the second file (flagged by >).

Except in rare circumstances, diff finds the smallest sufficient set of file differences.

OPTIONS

- b Ignore trailing blanks (spaces and tabs) and compare other strings of blanks as equal.
- e Produce a script of a, c, and d commands for the text editor ed which will recreate *file2* from *file1*. In connection with -e, the following Shell program may help maintain multiple versions of a file. Only an ancestral file (\$1) and a chain of version-to-version ed scripts (\$2,\$3,...) made by diff need be on hand. A “latest version” appears on the standard output. The Shell program is:

```
(shift; cat $*; echo '1,$p') | ed - $1
```
- f Produce a script similar to the one created by the -e option, but in the opposite order.
- h Produce a quick comparison of specified files or directories. Works only when changed stretches are short and well-separated, but does

work on files of unlimited length. Options `-e` and `-f` are unavailable with this option.

CAUTIONS

Editing scripts produced by using the `-e` or `-f` option are naive about creating lines consisting of a single period (.).

FILES

`/tmp/d?????`

`/usr/lib/diffh` (for `-h`)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, and 2 for trouble.

“Missing newline at end of file X” indicates that the last line of file X did not have a newline. If the lines are different, they will be flagged and output, although the output will seem to indicate they are the same.

RELATED INFORMATION

`cmp(1)`, `comm(1)`, `ed(1)`.

NAME

diff3 – 3-way differential file comparison

USAGE

diff3 [**-ex3**] *file1 file2 file3*

DESCRIPTION

Diff3 compares three versions of a file, and flags disagreeing text ranges with these codes:

```

=====      all three files are different
=====1     file1 is different
=====2     file2 is different
=====3     file3 is different

```

Diff3 indicates the type of change made in converting a given range of a given file to some other file in one of these ways:

```

f : n1 a      Text is to be appended after line number n1 in file f, where
                  f = 1, 2, or 3.

f : n1 , n2 c  Text is to be changed in the range line n1 to line n2. If n1
                  = n2, the range may be abbreviated to n1.

```

The original contents of the range immediately follow a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file are suppressed.

OPTIONS

```

-e          Publish a script for the text editor ed(1) that incorporates into file1 all
             changes between file2 and file3. These changes are normally flagged by
             ===== and =====3.

-x (-3)     Produce a script to incorporate only changes flagged ===== (=====3).
             The following command applies the resulting script to file1:

             (cat script; echo '1,$p') | ed - file1

```

CAUTIONS

Text lines consisting of a single period defeat the **-e** option.

Files longer than 64K bytes do not work.

FILES

*/tmp/d3**

/usr/lib/diff3prog

RELATED INFORMATION

diff(1).

NAME

diffmk – mark differences between files

USAGE

diffmk *name1 name2 name3*

DESCRIPTION

Diffmk is a shell procedure that compares two versions of a file and creates a third file that includes “change mark” commands for **nroff**(1) or **troff**(1). *Name1* and *name2* are the old and new versions of the file. **Diffmk** generates *name3*, which contains the lines of *name2* plus inserted formatter “change mark” (.mc) requests. When *name3* is formatted, changed or inserted text is shown by a pipe (|) at the right margin of each line. The position of deleted text is shown using a single asterisk (*).

If the pipe (|) and asterisk (*) characters are inappropriate, you may edit a copy of **diffmk** to change them.

EXAMPLE

To use **diffmk** as a means for producing listings of C (or other) programs with changes marked, type the following:

```
diffmk old.c new.c tmp; nroff macs
```

The file *macs* might contain this:

```
.pl 1  
.ll 77  
.nf  
.eo  
.nc
```

The .ll request might specify a different line length, depending on the nature of the program being printed. The .eo and .nc requests are probably needed only for C programs.

CAUTIONS

You may need to adjust some output manually. For example, file differences involving only formatting requests may produce undesirable output. Replacing one space with two produces a “change mark” on the preceding or following line of output.

RELATED INFORMATION

diff(1), **nroff**(1), **troff**(1).

NAME

dircmp – directory comparison

USAGE

dircmp [**-d**] [**-s**] [**-wn**] *dir1 dir2*

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. It generates listings of files unique to each directory for all the options. If no option is entered, **dircmp** outputs a list indicating whether the filenames common to both directories have the same contents.

OPTIONS

- d** Compare the contents of files with the same name in both directories, and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff(1)**.
- s** Suppress messages about identical files.
- wn** Change the width of the output line to *n* characters. The default width is 72.

RELATED INFORMATION

cmp(1), **diff(1)**.

NAME

du – summarize disk usage

USAGE

du [**-ars**] [*names*]

DESCRIPTION

Du prints the number of blocks (1024 bytes per block) contained in all files and directories specified by the *names* argument. The block count includes the indirect blocks of the file. A file with two or more links is only counted once. If the *names* argument is missing, a period (.) is used.

OPTIONS

- a** Generate an entry for each file.
- r** Generate messages about directories that cannot be read, files that cannot be opened, etc. **Du** is normally silent about such things.
- s** Print only the grand total of blocks for each of the specified *names*.

CAUTIONS

Absence of the **-a** or **-s** options causes an entry to be generated for each directory only.

If the **-a** option is not used, nondirectories given as arguments are not listed.

If too many distinct linked files exist, **du** counts the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

echo – echo arguments

USAGE

echo [*argument*] ...

DESCRIPTION

Echo writes each *argument*, separated by blanks and terminated by a newline, on the standard output. It also understands C-like escape conventions. Beware of conflicts with the shell's use of the backslash character (\):

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\n	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number <i>n</i> , which must start with a zero.

Echo is used to produce diagnostics in command files and for sending known data into a pipe.

RELATED INFORMATION

sh(1).

NAME

ed, red – text editor

USAGE

ed [-] [-p *string*] [*file*]

red [-] [-p *string*] [-x] [*file*]

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, ed simulates an e command (see below) on the named file, i.e, the file is read into the ed buffer so that it can be edited.

Ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of ed. It will only allow editing of files in the current directory. It prohibits executing Shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted Shell*).

The following size limitations apply to the text editor: 512 characters per line, 256 characters per global command list, 64 characters per filename, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes one word.

Commands to ed have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command requiring addresses has default addresses, so that the addresses can very often be omitted.

To understand addressing in ed it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command. The exact effect on the current line is discussed under the description of each command.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While ed is accepting text, it is in *input mode*. In this mode, ed recognizes *no* commands; it merely collects all input. Leave input mode by typing a period (.) alone at the beginning of a line.

If an interrupt signal (ASCII DEL or BREAK) is sent, ed prints a question mark (?) and returns to its command level.

Ed supports a limited form of *regular expression* notation. Regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE.

For more specific information concerning valid commands, as well as the construction of addresses and regular expressions used by the text editor, see the appropriate sections below.

OPTIONS

- Suppress the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*.
- p Specify the prompt to be equal to *string*.

REGULAR EXPRESSIONS

The following one-character regular expressions (REs) match a *single* character:

- 1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.
- 1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:
 - a. ., *, [, and \ (the period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]). Refer to 1.4 below.
 - b. ^ (caret), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]). Refer to 1.4 below.
 - c. \$ (dollar sign), which is special at the *end* of an entire RE (see 3.2 below).
 - d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)
- 1.3 A period (.) is a one-character RE that matches any character except newline.
- 1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a caret (^), the one-character RE matches any character *except* newline and the remaining characters in the string. The caret has this special meaning *only* if it occurs first in the string. The minus sign (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is

equivalent to [0123456789]. The minus sign loses this special meaning if it occurs first (after an initial caret, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial caret, if any); for example, []a-f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

Use the following rules to construct *REs* from one-character *REs*:

- 2.1 A one-character *RE* is a *RE* that matches whatever the one-character *RE* matches.
- 2.2 A one-character *RE* followed by an asterisk (*) is a *RE* that matches *zero* or more occurrences of the one-character *RE*. If there is any choice, the longest leftmost string that permits a match is chosen.
- 2.3 A one-character *RE* followed by $\backslash\{m\}$, $\backslash\{m,\}$, or $\backslash\{m,n\}$ is a *RE* that matches a *range* of occurrences of the one-character *RE*. The values of *m* and *n* must be non-negative integers less than 256; $\backslash\{m\}$ matches *exactly* *m* occurrences; $\backslash\{m,\}$ matches *at least* *m* occurrences; $\backslash\{m,n\}$ matches *any number* of occurrences *between* *m* and *n* inclusive. Whenever a choice exists, the *RE* matches as many occurrences as possible.
- 2.4 The concatenation of *REs* is a *RE* that matches the concatenation of the strings matched by each component of the *RE*.
- 2.5 A *RE* enclosed between the character sequences $\backslash($ and $\backslash)$ is a *RE* that matches whatever the unadorned *RE* matches.
- 2.6 The expression $\backslash n$ matches the same string of characters matched by an expression enclosed between $\backslash($ and $\backslash)$ *earlier* in the same *RE*. Here *n* is a digit; the subexpression specified is that beginning with the *n*-th occurrence of $\backslash($ counting from the left. For example, the expression $\backslash(.*\backslash)\backslash 1\$$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

- 3.1 A caret (^) at the beginning of an *entire RE* constrains that *RE* to match an *initial* segment of a line.
- 3.2 A dollar sign (\$) at the end of an *entire RE* constrains that *RE* to match a *final* segment of a line.

The construction $\backslash\textit{entire RE}\backslash$ constrains the *entire RE* to match the entire line. The null *RE* (e.g., $\backslash\backslash$) is equivalent to the last *RE* encountered.

ADDRESSES

Construct addresses using the following:

1. A period (.) addresses the current line.
2. A dollar sign (\$) addresses the last line of the buffer.
3. A decimal number n addresses the n -th line of the buffer.
4. An 'x addresses the line marked with the mark name character x , which must be a lowercase letter. Lines are marked with the k command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer, and stops at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (or minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean .-5.
9. If an address ends with + or -, then one (1) is added to or subtracted from the address, respectively. As a consequence of this rule and rule 8, the address of minus sign (-) refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the caret (^) in addresses is entirely equivalent to a minus. Moreover, trailing plus and minus characters have a cumulative effect, so that two minus signs together (- -) refer to the current line less two lines.
10. For convenience, a comma (,) stands for the address pair 1,\$, while a semicolon (;) stands for the pair .,\$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given. If more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,), but they can also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5 and 6). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

COMMANDS

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

Usually, only one command can appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by the *l*, *n*, or *p* commands, in which case the current line is either listed, numbered, or printed respectively.

If the closing delimiter of a RE or a replacement string (e.g., /) is the last character before a newline, you may omit that delimiter. *Ed* then prints the addressed line. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

(.)a
<text>
.

Read the given text and append it after the addressed line; leave . at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the “appended” text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.)c
<text>
.

Delete the addressed lines, then accept input text that replaces these lines; leave a period (.) at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

Delete the addressed lines from the buffer. The line after the last line deleted becomes the current line. If the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

Delete the entire contents of the buffer, and then read in the named file; set the current line (.) to the last line of the buffer. If no filename is given, use the currently-remembered filename, if any (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default filename in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest

of the line is taken to be a Shell command whose output is to be read. Such a Shell command is *not* remembered as the current filename. See `sh(1)` for more information about the Shell. Refer to *DIAGNOSTICS* below.

E *file*

Perform a function similar to the `e` command, except do not check to see whether any changes have been made to the buffer since the last `w` command.

f *file*

If *file* is given, change the currently-remembered filename to *file*; otherwise, print the currently-remembered filename.

(1,\$)g/*RE/command list*

First, mark every line that matches the given RE. Then, for every such line, execute the given *command list* with period (.) initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must end with a backslash (\); `a`, `i`, and `c` commands and associated input are permitted. The period (.) that terminates input mode may be omitted if it is the last line of the *command list*. An empty *command list* is equivalent to the `p` command. The `g`, `G`, `v`, and `V` commands are *not* permitted in the *command list*.

(1,\$)G/*RE/*

First, mark every line that matches the given RE. Then, for every such line, print that line, and change the current line pointer (.) to that line. Any *one* command (other than one of the `a`, `c`, `i`, `g`, `G`, `v`, and `V` commands) may then be input and executed. After the execution of that command, print the next marked line, and so on; a newline acts as a null command; an `&` causes the re-execution of the most recent command executed within the current invocation of `G`. Note that the commands input as part of the `G` command execution may address and affect *any* lines in the buffer. The `G` command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

Give a short error message that explains the reason for the most recent diagnostic that begins with a question mark (?).

H

Enter a mode in which error messages are printed for all subsequent diagnostics that begin with a question mark (?). Also, explain any such previous diagnostics. The `H` command alternately turns this mode on and off; it is initially off.

(.)i

<text>

Insert the given text before the addressed line; leave the current line pointer (.) at the last inserted line, or, if there were none, at the addressed line. This command differs from the `a` command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters

that may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)j

Join contiguous lines by removing the appropriate newline characters. If exactly one address is given, this command does nothing.

(.)kx

Mark the addressed line with name *x*, which must be a lowercase letter. The address '*x*' then addresses this line; the current line (.) is unchanged.

(.,.)l

Print the addressed lines in an unambiguous way. A few nonprinting characters (e.g., *tab*, *backspace*) are represented by mnemonic overstrikes. All other nonprinting characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)ma

Reposition the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a*, moving the addressed line(s) to the beginning of the file. It is an error if address *a* falls within the range of moved lines; the current line (.) becomes the last line moved.

(.,.)n

Print the addressed lines, preceding each line by its line number and a tab character. Leave the current line pointer (.) at the last line printed. The *n* command may be appended to any command other than *e*, *f*, *r*, or *w*.

(.,.)p

Print the addressed lines. Leave the current line pointer (.) at the last line printed. The *p* command may be appended to any command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

Prompt with an asterisk (*) for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

Exit the editor. No automatic write of a file is done.

Q

Exit the editor without checking to see whether changes have been made in the buffer since the last *w* command.

(\$)r *file*

Read in the given file after the addressed line. If no filename is given, use the currently-remembered filename, if any (see the *e* and *f* commands). Do not change the currently-remembered filename unless *file* is the very first filename mentioned since *ed* was invoked. Address 0 is legal for *r*, causing the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed and the current line is set to the last line read in. If *file* is replaced by an exclamation point (!), the rest of the line is taken to be a Shell command whose output is to be read. See *sh*(1) for more information about how the Shell operates. For example, *\$r !ls* appends the current

directory to the end of the file being edited. Such a Shell command is *not* remembered as the current filename.

(.,.)s/RE/replacement/ or
 (.,.)s/RE/replacement/g or
 (.,.)s/RE/replacement/n n = 1-512

Search each addressed line for an occurrence of the specified RE. In each line in which a match is found, replace all (nonoverlapped) matched strings with the *replacement* if the global replacement indicator *g* appears after the command. If the global indicator does not appear, replace only the first occurrence of the matched string. If a number *n* appears after the command, replace on the *n*th occurrence of the matched string on each addressed line. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or newline may be used instead of / to delimit the RE and the *replacement*; the current line becomes the last line on which a substitution occurred.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it with a backslash (\). As a more general feature, the characters \n, where *n* is a digit, are replaced by the text matched by the *n*th regular subexpression of the specified RE enclosed between \ (and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \ starting from the left. When the percent sign (%) is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The percent sign loses its special meaning when it is in a replacement string of more than one character or is preceded by a backslash (\).

A line may be split by substituting a newline character into it. The newline in the *replacement* must be escaped by preceding it by a backslash. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)ta

Perform the same functions as the *m* command, except place a *copy* of the addressed lines after address *a* (which may be 0). The current line becomes the last line of the copy.

u

Nullify the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,\$)v/RE/command list

Perform the same functions as the *g* command, but execute the *command list* with . initially set to every line that does *not* match the RE.

(1,\$)V/RE/

Perform the same functions as the **G** command, but mark only those lines that do *not* match the RE during the first step.

(1,\$)w file

Write the addressed lines into the named file. If the file does not exist, create it with mode 666 (readable and writable by everyone), unless the *umask* setting dictates otherwise. See **sh(1)** for more information about this. The currently-remembered filename is *not* changed unless *file* is the very first filename mentioned since **ed** was invoked. If no file name is given, the currently-remembered filename, if any, is used (see the **e** and **f** commands); the pointer to the current line (.) is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by **!**, the rest of the line is taken to be a Shell command whose standard input is the addressed lines. Such a Shell command is *not* remembered as the current filename.

(\$)=

Type the line number of the addressed line without changing the location of the current line.

!shell command

Send the remainder of the line after the exclamation point (!) to the UNIX system Shell, **sh (1)**, to be interpreted as a command. Within the text of that command, the unescaped percent character (%) is replaced with the remembered filename; if an exclamation point (!) appears as the first character of the Shell command, it is replaced with the text of the previous Shell command. Thus, two exclamation points in sequence (!!) will repeat the last Shell command. If any expansion is performed, the expanded line is echoed; the current line is left unchanged.

(.+1)<newline>

An address alone on a line causes the addressed line to be printed. A newline alone is equivalent to **+.1p**; it is useful for stepping forward through the buffer.

CAUTIONS

A **!** command cannot be subject to a **g** or **v** command.

The **!** command and the **!** escape from the **e**, **r**, and **w** commands cannot be used if the editor is invoked from a restricted Shell. See **sh(1)** for more information about the Shell.

The sequence **\n** in a regular expression does not match a newline character.

The **l** command mishandles DEL.

Characters are masked to seven bits on input.

If the editor input is coming from a command file (i.e., `ed file < ed-cmd-file`), the editor exits at the first failure of a command in the command file.

When reading a file, `ed` discards ASCII NUL characters and all characters after the last newline. Files containing characters not in the ASCII set (bit 8 on) cannot be edited using `ed`.

FILES

<code>/tmp/e#</code>	temporary; # is the process number
<code>ed.hup</code>	work is saved here if the terminal is hung up

DIAGNOSTICS

<code>?</code>	Command errors
<code>?file</code>	Inaccessible file

If you have made changes in the buffer since the last `w` command that wrote the entire buffer, `ed` questions attempts to destroy its buffer via the `e` or `q` commands. It prints a question mark before allowing you to continue editing. If you type a second `e` or `q` command at this point, these commands will take effect without further questioning from `ed`. The dash (`-`) command-line option inhibits this feature.

Use the `help` and `Help` commands for detailed explanations.

RELATED INFORMATION

`grep(1)`, `sed(1)`, `sh(1)`, `stty(1)`, `regex(5)`.

NAME

edit – text editor (a variant of **ex** for casual users)

USAGE

edit [**-r**] *file* ...

DESCRIPTION

Edit is a variant of the text editor **ex(1)** and is recommended if you are a new or casual user who desires to use a command-oriented editor.

If *file* already exists, **edit** will tell you how many lines and characters are in the file, before you begin editing. If the file does not yet exist, and you wish to create it as a result of the **edit** session, you must supply a name that does not match any of the other filenames in the current directory.

For more complete details on the use of **edit**, see the *DOMAIN/IX Text Processing Guide*.

DIAGNOSTICS

“No write since last change.”

Attempted to quit from the editor after modifying the buffer, without writing out the file.

Other diagnostics are usually self-explanatory.

RELATED INFORMATION

ex(1), **vi(1)**.

NAME

enable, disable – enable or disable LP printers

USAGE

enable *printers*

disable [-c] [-r[“*reason*”]] *printers*

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests from lp(1). Use lpstat(1) to find the status of *printers*.

Disable deactivates the named *printers*, preventing them from printing requests from lp(1). By default, any requests that are currently printing on the designated *printers* will be reprinted in their entirety either on the same *printer* (when it is re-enabled) or on another member of the same class of *printer*. Use lpstat(1) to find the status of *printers*.

OPTIONS

These options apply only to **disable**.

-c Cancel any requests that are currently printing on any of the designated *printers*.

[-r[“*reason*”]]

Give a *reason* why the *printers* are disabled. (The reason must appear in double quotes.) This reason applies to all *printers* mentioned up to the next -r option. If the -r option is not present or the -r option is given without a *reason*, then the system supplies a default message that will be printed, usually, “reason unknown”. *Reason* is reported by lpstat(1).

FILES

/usr/spool/lp/*

spooling directories

RELATED INFORMATION

lp(1)

lpstat(1)

NAME

env – set environment for command execution

USAGE

env [-] [*name=value*] ... [*command arguments*]

DESCRIPTION

Env obtains the current environment, modifies it according to its *arguments*, then executes the command with the modified environment. Arguments of the form *name = value* are merged into the inherited environment before the command is executed. A dash (–) at the beginning of the command line causes **env** to ignore the inherited environment completely, executing the command with the environment specified by the *arguments*.

If you do not supply a command, **env** prints the resulting environment with one name-value pair per line.

RELATED INFORMATION

sh(1), **exec**(2), **environ**(5).

NAME

eqn, neqn, checkeq – format mathematical text for **nroff** or **troff**

USAGE

eqn [**-dxy**] [**-pn**] [**-sn**] [**-fn**] [*files*]

neqn [**-dxy**] [**-pn**] [**-sn**] [**-fn**] [*files*]

checkeq [*files*]

DESCRIPTION

Eqn is a **troff(1)** preprocessor for typesetting mathematical text on a phototypesetter. **Neqn** is used for the same purpose with **nroff(1)** on typewriter-like terminals. These commands are normally used in this or a similar manner:

```
eqn files | troff
neqn files | nroff
```

If no files are specified, **eqn** and **neqn** read from the standard input. A line beginning with **.EQ** marks the start of an equation; a line beginning with **.EN** marks the end of an equation. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as delimiters. Subsequent text between delimiters is then treated as **eqn** input. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is not between delimiters or between **.EQ** and **.EN** passes through untouched.

Checkeq reports missing or unbalanced delimiters and **.EQ/.EN** pairs.

Tokens within **eqn** are separated by spaces, tabs, newlines, braces, double quotes, tildes, and carets. Braces (**{ }**) are used for grouping. Generally speaking, anywhere a single character such as *x* can appear, a complicated construction enclosed in braces may be used instead. A tilde (**~**) represents a full space in the output, a caret (**^**) half as much.

DOMAIN/IX supports the UNIX System III version of **eqn**, **neqn**, and **checkeq**.

OPTIONS

- dxy** Set delimiters to characters *x* and *y*. More commonly done with **delim xy** between **.EQ** and **.EN**.
- pn** Change the normal reduction of three points from the previous size of subscripts and superscripts to the value of *n*.
- sn** Change point size to the value of *n*.
- fn.** Change font to the value of *n*.

Keywords such as **sum** (Σ), **int** (\int), **inf** (∞), and shorthands such as \geq (\geq), \neq (\neq), and \rightarrow (\rightarrow) are recognized. Greek letters are spelled out in the desired case, as in *alpha* (α), or **GAMMA** (Γ). Mathematical words such as **sin**, **cos**, and **log** are made roman automatically. **Troff**(1) four-character escapes such as $\backslash dd$ (\ddagger) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched. This permits keywords to be entered as text, and can be used to communicate with **troff** when all else fails.

For full details, see the *DOMAIN/IX Text Processing Guide*.

RELATED INFORMATION

mm(1)

mmt(1)

tbl(1)

troff(1)

DOMAIN/IX Text Processing Guide

NAME

ex – text editor

USAGE

ex [-] [-v] [-t *tag*] [-r *file*] [-R] [+*command*] [-l] *file* ...

DESCRIPTION

Ex is the root of a family of editors, which also includes `edit(1)` and `vi(1)`. Ex is a superset of `ed(1)`, with the most notable extension being a display editing facility. Display-based editing is the focus of `vi`.

Full details concerning the use of `ex` are explained in the *DOMAIN/IX Text Processing Guide*.

OPTIONS

- Suppress all interactive-user feedback. This is useful in processing editor scripts.
- v Invoke `vi(1)`.
- t *tag* Edit the file containing the *tag* and position the editor at its definition.
- r *file* Recover *file* after an editor or system crash. If you do not specify *file*, `ex` prints a list of all saved files.
- R Set mode to *readonly* to prevent accidental overwriting of the file.
- +*command* Begin editing by executing the specified editor search or positioning *command*.
- l Set *lisp* mode to indent appropriately for LISP code. Modify the parentheses, braces, and left and right double brackets commands, i.e., `()`, `{ }`, `[[`, and `]]`, in `vi(1)` to have meaning for LISP.

CAUTIONS

The `undo` command causes all marks to be lost on lines changed and then restored if the marked lines were changed. It also never clears the buffer-modified condition.

The `z` command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the dash (`-`) option is used on the command line.

There is no easy way to do a single scan ignoring case.

No warning messages appear when text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

FILES

<i>/usr/lib/ex?.?recover</i>	recover command
<i>/usr/lib/ex?.?preserve</i>	preserve command
<i>/usr/lib/*/*</i>	capabilities of terminals
<i>\$HOME/.exrc</i>	editor start-up file

awk(1), ed(1), grep(1), sed(1), vi(1).

NAME

expr – evaluate arguments as an expression

USAGE

expr arguments

DESCRIPTION

Expr takes the *arguments* supplied as an expression, evaluates them, and writes the results on the standard output. Blank spaces must separate terms of the expression, and characters special to the Shell must be escaped. Note that 0 is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, two's-complement numbers.

OPERATORS AND KEYWORDS

Expr uses the operators and keywords listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*

Return the first *expr* if it is neither null nor zero. Otherwise, return the second *expr*.

expr \& *expr*

Return the first *expr* if neither *expr* is null or zero, otherwise return 0.

expr { =, >, >=, <, <=, != } *expr*

Return the result of an integer comparison if both arguments are integers, Otherwise, return the result of a lexical comparison.

expr { +, - } *expr*

Add or subtract integer-valued arguments.

expr { *, /, } *expr*

Multiply, divide, or remainder the integer-valued arguments.

expr : *expr*

Compare the first argument with the second argument (which must be a regular expression). Regular expression syntax is the same as that of *ed*(1), except that all patterns are “anchored,” that is, begin with a caret (^). Therefore, in this context, the caret is not a special character. Normally, the matching operator returns the number of characters matched (0 on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

EXAMPLES

To add 1 to the Shell variable *a*, type the following:

```
a='expr $a + 1'
```

To return the last segment of a pathname (i.e., file), use this:

```
# 'For $a equal to either  
expr //$a : '.*\(.*\)'
```

To return the number of characters in \$VAR, execute this command:

```
expr $VAR : '.*'
```

CAUTIONS

After argument processing by the Shell, `expr` cannot tell the difference between an operator and an operand, except by the value. If \$a is an equal sign (=), the command:

```
expr $a = =
```

looks like:

```
expr = = =
```

as the arguments are passed to `expr` (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

EXIT CODE

As a side effect of expression evaluation, `expr` returns the following exit values:

- 0 expression is neither null nor zero
- 1 expression is null or zero
- 2 expression is invalid

DIAGNOSTICS

"syntax error"	Operator/operand error.
"non-numeric argument"	Arithmetic attempted on a non-numeric string.

RELATED INFORMATION

ed(1), sh(1).

NAME

factor – factor a number

USAGE

factor [*number*]

DESCRIPTION

When **factor** is invoked with a *number* argument, it prints each argument's prime factors the proper number of times. The *number* specified must be a positive integer less than 2^{56} (about 7.2 times 10^{16}).

Without an argument, **factor** waits for a number to be typed in. After printing the number's prime factors, it waits for another number.

Factor exits if it encounters a zero or any non-numeric character. The maximum time to factor is proportional to \sqrt{n} and occurs when n is prime or the square of a prime.

DIAGNOSTICS

“Ouch” for input out of range or for garbage input.

NAME

file – determine file type

USAGE

file [**-c**] [**-f** *ffile*] [**-m** *mfile*] *arg* ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, **file** examines the first 512 bytes and tries to guess its language. If an argument is an executable a.out file, **file** prints the version stamp, provided it is greater than zero. See **ld(1)** for more information about a.out files.

OPTIONS

- f** *ffile* Take the next argument to be a file containing the names of the files to be examined. **File** uses the file */etc/magic* to identify files that have some sort of *magic number*; that is, any file containing a numeric or string constant that indicates its type. The commentary at the beginning of */etc/magic* explains its format.
- m** *mfile* Use an alternate magic file.
- c** Check the magic file for format errors. Do no file typing.

FILES

/etc/magic

RELATED INFORMATION

ld(1).

NAME

find – find files

USAGE

find *path-name-list expression*

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the *path-name-list*, seeking files that match a Boolean *expression* written in the primaries given below. The DOMAIN/IX implementation of **find** does not follow links.

EXPRESSIONS

(In the descriptions below, the argument *n* is used as a decimal integer where *+n* means more than *n*, *-n* means less than *n*, and *n* means exactly *n*).

- name *file*** True if *file* matches the current filename. Normal Shell argument syntax may be used if escaped, but watch out for [, ? and *.
- perm *onum*** True if the file permission flags exactly match the octal number *onum*. Information about file permissions is found in **chmod**(1). If *onum* is prefixed by a minus sign, more flag bits (017777) become significant and the flags are compared. See **stat**(2) for details.
- type *c*** True if the type of the file is *c*, where *c* is *b* (block special file), *c* (character special file), *d* (directory), *p* (FIFO, or named pipe), *f* (plain file), or *l* (softlink).
- links *n*** True if the file has *n* links.
- user *uname*** True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a log-in name in the */etc/passwd* file, it is taken as a user ID.
- group *gname*** True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the */etc/group* file, it is taken as a group ID.
- size *n*[*c*]** True if the file is *n* blocks long (1024 bytes per block). If *n* is followed by a *c*, the size is in characters.
- atime *n*** True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by **find** itself.
- mtime *n*** True if the file has been modified in *n* days.
- ctime *n*** True if the file has been changed in *n* days.
- exec *cmd*** True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument { } is replaced by the current pathname.

-ok <i>cmd</i>	Like -exec , except this prints the generated command line with a question mark first, and executes only if you respond by typing y .
-print	Always true; print the current pathname.
-cpio <i>device</i>	Always true; write the current file on <i>device</i> in cpio(4) format (5120-byte records).
-newer <i>file</i>	True if the current file has been modified more recently than the argument <i>file</i> .
-depth	Always true; descend the directory hierarchy so that all entries in a directory are acted on before the directory itself. Can be useful when find is used with cpio(1) to transfer files contained in directories without write permission.
(<i>expression</i>)	True if a parenthetical expression is true (parentheses are special to the Shell and must be escaped).

OPERATORS

The primaries listed above may be combined using the following operators (in order of decreasing precedence):

- 1) The negation of a primary (**!** is the unary *not* operator).
- 2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
- 3) Alternation of primaries (**-o** is the *or* operator).

EXAMPLE

To remove all files named *a.out* or **.o* that have not been accessed for a week:

```
# find / ( -name a.out -o -name '*.o' )-atime +7 -exec rm {} ;
```

FILES

/etc/passwd
/etc/group

RELATED INFORMATION

chmod(1), **cpio(1)**, **sh(1)**, **test(1)**, **stat(2)**, **cpio(4)**.

NAME

fsplit – split FORTRAN or ratfor files

USAGE

fsplit *options files*

DESCRIPTION

The **fsplit** command splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Normally, procedure *X* is put in file *X.f* or *X.r*, depending on the language option chosen. The following exceptions apply: *main* is put in the file *MAIN.[fr]*, and unnamed *blockdata* segments in the files *blockdataN.[fr]* (where *N* is a unique integer value for each file).

OPTIONS

- f** Use FORTRAN source program files as input.
- r** Use ratfor(1) source program files as input.
- s** Strip FORTRAN input lines to 72 or fewer characters with trailing blanks removed.

RELATED INFORMATION

csplit(1)
ratfor(1)
split(1)

NAME

ad – graphical device routines and filters

USAGE

ad [- options] [GPS file(s)]

DESCRIPTION

The command described below resides in */usr/bin/graf* (see *graphics(1G)*).

ad Output is GMR subroutine calls to the Apollo node. A viewing window is computed from the maximum and minimum points from all input file(s) unless options are given. If no file is given, the standard input is assumed. If the options **p** or **w** are not given the graphics will be displayed in the current pad with a DIALOG interface. This allows you to zoom in on particular portions of the GPS output. Options are:

rn Window on GPS region *n*, *n* between 1 and 25 inclusive.

pprinter

Plot the GPS output on the named *printer*.

w [*h:w:t:l*] Plot the GPS output in a new pad of height *h*, width *w*, and top left corner at (*t,l*). If no values are specified the new pad will be the same size as the current pad. If height is specified but not width the width is assumed equal to height.

n Do not interpret line weights.

EXAMPLES

Display A.g and B.g. The viewing window is built to include all A.g and B.g.

ad A.g B.g

Display A,g in a new pad with a width and height of 500 pixels.

ad -w500 A.g

Send A.g to a printer named cx, via a bitmap of 1024 by 1024 pixels.

RELATED INFORMATION

ged(1G), graphics(1G)

NAME

`get` – get a version of an SCCS file

USAGE

```
get [ -rSID ] [ -ccutoff ] [ -ilist ] [ -xlist ] [ -wstring ] [ -aseq-no. ] [ -k ] [ -e ] [
-l [ p ] ] [ -p ] [ -m ] [ -n ] [ -s ] [ -b ] [ -g ] [ -t ] file ...
```

DESCRIPTION

`Get` generates an ASCII text file from each named SCCS file according to the options that you specify in the command line. These options may appear in any order, but they apply to all named SCCS files. If you supply a directory name, `get` behaves as though each file in the directory is specified as a named file, except that it silently ignores non-SCCS and unreadable files. If you specify a dash (-) for the name of *file*, `get` reads the standard input, taking each line to be the name of an SCCS file to be processed.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS filename by simply removing the “s.” prefix.

Each of the options is explained below as though only one SCCS file is to be processed, but the effects of any option applies independently to each named file.

For each file processed, `get` responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

OPTIONS

-rSID Specify the SCCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. See Table 1 to determine what version of an SCCS file is retrieved as a function of the SID specified.

-ccutoff Indicate *cutoff* date-time in this format:

YY[MM[DD[HH[MM[SS]]]]]

Include, in the ASCII text file, only deltas to the SCCS that were produced up to the specified *cutoff* date-time. Units omitted from the date-time default to their maximum possible values; that is, `-c7502` is equivalent to `-c750228235959`. Any number of non-numeric characters may separate the various two-digit pieces of the *cutoff* date-time. Thus, you can specify a *cutoff* date in the following format:

`-c77/2/2 9:22:25.`

You can use the E and U identification keywords for nested executions of `get` within the command line input, as in this example:

```
~!get -cE U s.file
```

- e Edit or make a change (delta) to the SCCS file via a subsequent use of `delta(1)`. The SID of the delta to be made appears after the SID accessed, and before the number of lines generated. If you have more than one named file, or if you specify a directory or standard input, `get` prints each filename (preceded by a new-line) before processing it. For a particular version SID of the SCCS file, using this option prevents further executions of `get` for editing on the same SID until `delta(1)` is executed or the `j` (joint edit) flag is set in the SCCS file. Concurrent use of this option for different SIDs is always allowed.

If the *g-file* generated by `get` with an `-e` option is accidentally ruined in the process of editing it, you can generate it by re-executing the `get` command with the `-k` option in place of the `-e` option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file is enforced when the `-e` option is used. See `admin(1)` for more information.

- b Provide the new delta with an SID in a new branch as shown in Table 1. This option only works when it is used with the `-e` option, the `b` flag is present in the named file, and the retrieved delta is a leaf delta (one that has no successors on the SCCS file tree). A branch delta may always be created from a non-leaf delta. Refer to `admin(1)` for further details.
- i*list* Specify a *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

`<list> ::= <range> | <list> , <range>`
`<range> ::= SID | SID - SID`

SID, the SCCS IDentification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1. Included deltas are identified on the list by the notation "Included".

- x*list* Specify a *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. Excluded deltas are listed following the notation "Excluded". See the `-i` option for the *list* format.
- k Suppress replacement of identification keywords (see below) in the retrieved text by their value. The `-k` option is implied by the `-e` option.
- l [*p*] Write a delta summary into an *l-file*. If `-lp` is used, do not create an *l-file*. Instead, write the delta summary on the standard output. See *FILES* for the format of the *l-file*.
- p Write the text retrieved from the SCCS file on the standard output. Do not create a *g-file*. Send all output that normally goes to the standard output

to file descriptor 2 instead, unless the `-s` option is used, in which case remove the output.

- `-s` Suppress all output normally written on the standard output. However, do not alter fatal error messages, which always go to file descriptor 2.
- `-m` Precede each text line retrieved from the SCCS file by the SID of the delta that inserted the text line in the SCCS file. The format is as follows: SID, followed by a horizontal tab, followed by the text line.
- `-n` Precede each generated text line with the M identification keyword value (see below). The format is as follows: M value, followed by a horizontal tab, followed by the text line. When both the `-m` and `-n` options are used, the format is: M value, followed by a horizontal tab, followed by the `-m` option generated format.
- `-g` Suppress the actual retrieval of text from the SCCS file. Used primarily to generate an *l-file* or to verify the existence of a particular SID.
- `-t` Access the most recently created ("top") delta in a given release (e.g., `-r1`), or release and level (e.g., `-r1.2`).
- `-w string` Substitute *string* for all occurrences of `@(#)get.1` when executing a `get` on the named file.
- `-aseq-no.` Specify the delta sequence number of the SCCS file delta (version) to be retrieved. See `sccsfile(4)` for further details. This option is used by the `comb(1)` command; it is not a particularly useful option. If both the `-r` and `-a` options are specified, the `-a` option is used. Be careful when using the `-a` option in conjunction with the `-e` option, as the SID of the delta to be created may not be the one that you had expected. The `-r` option can be used with the `-a` and `-e` options to control the naming of the SID of the delta to be created.

Determination of SCCS Identification String

SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	—	R < mR and R does <i>not</i> exist	hR.mL**	hR.mL.(mB+1).1
R	—	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	—	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	—	Branch succ.	R.L.B.S	R.L.(mB+1).1

* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. For example, "R.mL" means the maximum level number within release R; "R.L.(mB+1).1" means the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R. If the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** An hR is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

Successor.

† The -b option is effective only if the b flag is present in the file. See `admin(1)` for further information. A dash by itself (—) means irrelevant.

‡ This case applies if the d (default SID) flag is *not* present in the file. If the d flag *is* present in the file, then the SID obtained from the d flag is

interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

<i>Keyword</i>	<i>Value</i>
M	Module name: either the value of the m flag in the file, or if absent, the name of the SCCS file with the leading s. removed.
I	SCCS Identification (SID) (R.L.B.S) of the retrieved text.
R	Release.
L	Level.
B	Branch.
S	Sequence.
D	Current date (YY/MM/DD).
H	Current date (MM/DD/YY).
T	Current time (HH:MM:SS).
E	Date newest applied delta was created (YY/MM/DD).
G	Date newest applied delta was created (MM/DD/YY).
U	Time newest applied delta was created (HH:MM:SS).
Y	Module type: value of the t flag in the SCCS file. See admin(1) .
F	SCCS filename.
P	Fully-qualified SCCS filename.
Q	The value of the q flag in the file. See admin(1) .
C	Current line number. This keyword is meant to identify program output messages such as "this should not have happened" when an error occurs. It should not be used on every line to provide sequence numbers.
Z	The four-character string @(#) recognizable by what(1) .
W	A shorthand notation for constructing what(1) strings for UNIX system program files. The format is: W~ = ~ZM<horizontal-tab>I
A	Another shorthand notation for constructing what(1) strings for non-UNIX system program files. The format is: A~ = ~ZY~M~IZ

FILES

Get may create several auxiliary files. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary filename is formed from the SCCS filename. The last component of all SCCS filenames must have the form *s.module-name*. The auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, the auxiliary file names of *s.xyz.c* are

xyz.c, l.xyz.c, p.xyz.c, and z.xyz.c, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the *-p* option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the *-k* option is used or implied its mode is 644; otherwise, its mode is 444. Only the real user needs write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the *-l* option is used; its mode is 444 and it is owned by the real user. Only the real user needs write permission in the current directory.

Lines in the *l-file* have the following format:

- a. A blank character if the delta was applied; * otherwise.
- b. A blank character if the delta was applied or was not applied and ignored; * if the delta was not applied and was not ignored.
- c. A code indicating a special reason why the delta was or was not applied. The codes are:
 - I: Included.
 - X: Excluded.
 - C: Cut off (by a *-c* option).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD~HH:MM:SS) of creation.
- h. Blank.
- i. Log-in name of the person who created the delta.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* passes information resulting from a *get* with an *-e* option along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an *-e* option for the same SID until *delta* is executed or the joint edit flag, *j*, is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file. The effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the log-in name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the *-i* option if it was present, followed by a blank and the *-x* option if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (two bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

CAUTIONS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the *-e* option is used.

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

- admin(1)**
- delta(1)**
- help(1)**
- prs(1)**
- what(1)**
- sccsfile(4)**

NAME

getopt – parse command options

USAGE

set -- ‘getopt *optstring* **’**

DESCRIPTION

Getopt breaks up options in command lines for easy parsing by shell procedures, and for checking for legal options. *Optstring* is a string of recognized option letters. See **getopt(3C)** for more information. If a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space.

The special option represented by two subsequent dashes (--) delimits the end of the options. If you use it explicitly, **getopt** recognizes it; otherwise, **getopt** generates it. In either case, this special option is placed at the end of the list of options.

The shell's positional parameters (\$1 \$2 ...) are reset so that each option is preceded by a dash (-) and is in its own positional parameter. Each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how you might process the arguments for a command that can take the options a or b, as well as the option o, which requires an argument:

```
set -- 'getopt abo: *'
if [ $? != 0 ]
then
    echo $USAGE
    exit 2
fi
for i in $*
do
    case $i in
        -a | -b)    FLAG=$i; shift;;
        -o)        OARG=$2; shift 2;;
        - -)       shift; break;;
        esac
    done
```

This code accepts any of the following as equivalent:


```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg - - file file
```

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

RELATED INFORMATION

sh(1), getopt(3C).

NAME

graph – draw a graph

USAGE

graph [*options*]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the **tplot(1G)**. filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ("), in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

- a** Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **-x**).
- b** Break (disconnect) the graph after each label in the input.
- c** Character string given by next argument is default label for each point.
- g** Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
- l** Next argument is label for graph.
- m** Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the TEKTRONIX 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).
- s** Save screen, do not erase before plotting.
- x [l]** If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.
- y [l]** Similarly for y.
- h** Next argument is fraction of space for height.
- w** Similarly for width.
- r** Next argument is fraction of space to move right before plotting.
- u** Similarly to move up before plotting.
- t** Transpose horizontal and vertical axes. (Option **-x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the `-s` option is present.
If a specified lower limit exceeds the upper limit, the axis is reversed.

RELATED INFORMATION

`graphics(1G)`, `spline(1G)`, `tplot(1G)`.

NOTES

Graph stores all points internally and drops those for which there is no room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

graphics – access graphical and numerical commands

USAGE

graphics [*-r*]

DESCRIPTION

Graphics prefixes the path name */usr/bin/graf* to the current **\$PATH** value, changes the primary shell prompt to *^*, and executes a new shell. The directory */usr/bin/graf* contains all of the Graphics subsystem commands. If the *-r* option is given, access to the graphical commands is created in a restricted environment; that is, **\$PATH** is set to *:/usr/bin/graf:/rbin:/usr/rbin* and the restricted shell, *rsh*, is invoked. To restore the environment that existed prior to issuing the **graphics** command, type EOT (control-d on most terminals). To logoff from the graphics environment, type **quit**.

The command line format for a command in **graphics** is *command name* followed by *argument(s)*. An argument may be a file name or an option string. A file name is the name of any DOMAIN/IX system file except those beginning with *-*. The file name *-* is the name for the standard input. An option string consists of *-* followed by one or more *option(s)*. An option consists of a keyletter possibly followed by a value. Options may be separated by commas.

The graphical commands have been partitioned into four groups.

Commands that manipulate and plot numerical data; see **stat(1G)**.

Commands that generate tables of contents; see **toc(1G)**.

Commands that interact with graphical devices; see **gdev(1G)**.

A collection of graphical utility commands; see **gutil(1G)**.

A list of the **graphics** commands can be generated by typing **whatis** in the **graphics** environment.

RELATED INFORMATION

gdev(1G), **gutil(1G)**, **stat(1G)**, **toc(1G)**.

gps(4) in the *UNIX System Programmer Reference Manual*.

UNIX System Graphics Guide.

NAME

grep, egrep, fgrep – search a file for a pattern

USAGE

grep [*options*] *expression* [*files*]

egrep [*options*] [*expression*] [*files*]

fgrep [*options*] [*strings*] [*files*]

DESCRIPTION

Commands of the **grep** family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. Patterns used by **grep** are limited regular *expressions* in the style of **ed**(1), using a compact nondeterministic algorithm. Those used by **egrep** are full regular *expressions*, using a fast deterministic algorithm that sometimes needs exponential space. Patterns used by **fgrep** are fixed *strings* that are fast and compact.

In all cases, the filename is output if there is more than one input file. Be careful, however, when using any of the following characters in *expression*: \$, *, [, ^, |, (,), and \. These characters are also meaningful to the shell. It is safest to enclose an entire *expression* argument in single quotes ('...').

Fgrep searches for lines containing one of the *strings* separated by newlines.

Egrep accepts regular expressions as in **ed**(1), except for \ and \, with the addition of the following rules:

1. A regular expression followed by a plus sign (+) matches one or more occurrences of the regular expression.
2. A regular expression followed by a question mark (?) matches zero or one occurrences of the regular expression.
3. Two regular expressions separated by a pipe character (|) or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is as follows: brackets ([]), asterisk (*), question mark (?), plus sign (+), concatenation, pipe (|), and newline.

OPTIONS

- | | |
|----|---|
| -v | Print all lines except those that match. |
| -x | Print only lines exactly matched (fgrep only). |
| -c | Print only a count of matching lines. |
| -i | Ignore upper- and lowercase distinction during comparisons. |
| -l | List only the filenames with matching lines. Separate the names |

with new-lines.

- n** Precede each line by its relative line number in the file.
- b** Precede each line by the block number (1024 bytes per block) on which it was found. Useful in locating disk block numbers by context.
- s** Suppress the error messages produced for nonexistent or unreadable files (**grep** only).
- e *expression*** Same as a simple *expression* argument, but useful when the *expression* begins with a dash (-). This option does not work with **grep**.
- f *file*** Take the regular *expression* (**egrep**) or *strings* list (**fgrep**) from the *file*.

CAUTIONS

Lines are limited to BUFSIZ (1024) characters (defined in */usr/include/stdio.h*); longer lines are truncated.

Egrep does not recognize ranges, such as *[a-z]*, in character classes.

If a line is embedded with nulls, **grep** only matches up to the first null; if the null matches, **grep** prints the entire line.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, and 2 for syntax errors or inaccessible files (even if matches were found).

RELATED INFORMATION

ed(1), **sed**(1), **sh**(1).

NAME

gutil – graphical utilities

USAGE

command-name [*options*] [*files*]

DESCRIPTION

Below is a list of miscellaneous device independent utility commands found in */usr/bin/graf*. If no files are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see *gps(4)*.

bel – send bel character to terminal

cvrtopt [=*sstring fstring istring tstring*] [*args*] – options converter
Cvrtopt reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a file name (a string not beginning with a *-*, or a *-* by itself) or an option string (a string of options beginning with a *-*). Output is of the form:

-option -option . . . file name(s)

All options appear singularly and preceding any file names. Options that take values (e.g., *-r1.1*) or are two-letters long must be described through options to *cvrtopt*.

Cvrtopt is usually used with *set* in the following manner as the first line of a shell procedure:

set - `cvrtopt =[options] \$@`

Options to *cvrtopt* are:

sstring *String* accepts string values.

fstring *String* accepts floating point numbers as values.

istring *String* accepts integers as values.

tstring *String* is a two-letter option name that takes no value.

String is a one- or two-letter option name.

gd [GPS *files*] – GPS dump
Gd prints a human readable listing of GPS.

gtop [*-rn u*] [GPS *files*] – GPS to *plot(4)* filter
Gtop transforms a GPS into *plot(4)* commands displayable by *plot* filters. GPS objects are translated if they fall within the window that circumscribes the first file unless an option is given.
 Options:

rn translate objects in GPS region *n*.

u translate all objects in the GPS universe.

pd [*plot(5) files*] – **plot(4)** dump
Pd prints a human readable listing of **plot(4)** format graphical commands.

ptog [*plot(5) files*] – **plot(4)** to GPS filter
Ptog transforms **plot(4)** commands into a GPS.

quit – terminate session

remcom [*files*] – remove comments
Remcom copies its input to its output with comments removed. Comments are as defined in C (i.e., /* comment */).

whatis [-o] [*names*] – brief on-line documentation
Whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command *whatis ** prints out every description.
Option:

o just print command options

yoo *file* – pipe fitting
Yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without *yoo*, this is not usually successful as it causes a read and write on the same file simultaneously.

RELATED INFORMATION

graphics(1G).

gps(4), *plot(4)* in the *UNIX System Programmer Reference Manual*.

NAME

help – ask for SCCS help

USAGE

help [*arguments*]

DESCRIPTION

Help finds information to explain the use of, or a particular message from, an SCCS command.

The arguments may be either message numbers (which normally appear in parentheses following messages), or command names, or one of the following types:

- | | |
|--------|--|
| type 1 | Begins with non-numeric, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines that produced the message. For example, <i>ge3</i> means message 3 from the <i>get(1)</i> command. |
| type 2 | Does not contain numerics, i.e., appears as a command name such as the name <i>get</i> . |
| type 3 | Is all numeric, e.g., 26. |

When all else fails, try **help stuck**.

FILES

<i>/usr/lib/help</i>	directory containing files of message text
<i>/usr/lib/help/helploc</i>	file containing locations of help files not in <i>/usr/lib/help</i>

DIAGNOSTICS

Use **help(1)** for explanations.

NAME

hyphen – find hyphenated words

USAGE

hyphen [*files*]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, **hyphen** may be used as a filter.

EXAMPLE

The following allows the proofreading of **nroffR(1)** hyphenation in *textfile*.

```
# mm textfile | hyphen
```

CAUTIONS

Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words. It often misses them completely or mangles them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

RELATED INFORMATION

mm(1), **nroff(1)**.

NAME

id – print user and group IDs and names

USAGE

id

DESCRIPTION

Id writes a message on the standard output, giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

RELATED INFORMATION

logname(1), getuid(2).

NAME

ipcrm – remove a message queue, semaphore set, or shared memory ID

USAGE

ipcrm [*options*]

DESCRIPTION

Ipcrm removes one or more specified message queue, semaphore, or shared memory identifiers. The manual pages for **msgctl(2)**, **semctl(2)**, and **shmctl(2)** describe the removals in detail. Use **ipcs(1)** to find the identifiers and keys.

OPTIONS

- q *msqid*** Remove the message queue identifier *msqid* from the system. Destroy the message queue and data structure associated with it.
- m *shmid*** Remove the shared memory identifier *shmid* from the system. After the last detach, destroy the shared memory segment and data structure associated with it.
- s *semid*** Remove the semaphore identifier *semid* from the system. Destroy the set of semaphores and data structure associated with it.
- Q *msgkey*** Remove from the system the message queue identifier created with the key *msgkey*. Destroy the message queue and data structure associated with it.
- M *shmkey*** Remove from the system the shared memory identifier created with the key *shmkey*. After the last detach, destroy the shared memory segment and data structure associated with it.
- S *semkey*** Remove from the system the semaphore identifier created with the key *semkey*. Destroy the set of semaphores and data structure associated with it.

RELATED INFORMATION

ipcs(1), **msgctl(2)**, **msgget(2)**, **msgop(2)**, **semctl(2)**, **semget(2)**, **semop(2)**, **shmctl(2)**, **shmget(2)**, **shmop(2)**.

NAME

ipcs – report interprocess communication facilities status

USAGE

ipcs [*options*]

DESCRIPTION

Ipcs prints certain information about active interprocess communication facilities, depending on specified options. By default, information is printed in short format for message queues, shared memory, and semaphores currently active in the system. Unless you specify a **-q**, **-m**, or **-s** option, **ipcs** prints information about all three types of facilities.

OPTIONS

- q** Print information about active message queues.
- m** Print information about active shared memory segments.
- s** Print information about active semaphores.
- b** Print information about the largest allowable size for the facility. This includes the maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores. See below for the meaning of columns in a listing.
- c** Print creator's log-in name and group name. See below.
- o** Print information on outstanding usage. This includes the number of messages on queue and total number of bytes in messages on queue for message queues, and the number of processes attached to shared memory segments.
- p** Print process number information. This includes the process ID of the following: last process to send a message, last process to receive a message on message queues, creating process, and last process to attach or detach on shared memory segments. See below.
- t** Print time of the last control operation that changed the access permissions for all facilities, last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, and last *semop(2)* on semaphores. See below.
- a** Print all information individually available through the **-b**, **-c**, **-o**, **-p**, and **-t** options.

OUTPUT INTERPRETATION

The column headings and the meaning of the columns in an *ipcs* listing are given below. The letters in parentheses indicate the *options* that cause the corresponding heading to appear; *all* means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do not determine which facilities are listed.

T	(all)	Type of facility: q message queue m shared memory segment s semaphore
ID	(all)	Identifier for the facility entry.
KEY	(all)	Key is used as an argument to <i>msgget</i> , <i>semget</i> , or <i>shmget</i> to create the facility entry. (Note: The key of a shared memory segment is changed to <i>IPC_PRIVATE</i> when the segment has been removed until all processes attached to the segment detach it.)
MODE	(all)	Facility access modes and flags. The mode consists of 11 characters in all.

The first two characters may be one each from the following sets, or one of the characters followed by a dash (-) (to signify that the corresponding special flag was not set), or a double dash (--).

The first set describes information about message queues:

- R** process is waiting on a *msgrcv*
- S** process is waiting on a *msgsnd*

The second set describes information about shared memory segments:

- D** associated shared memory segment has been removed (i.e., will disappear when the last process attached to the segment detaches it)
- C** associated shared memory segment is cleared when the first attach is executed

The next nine characters describe various types of permissions. They are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character shows permission to read, and the second character shows permission to write or alter the facility entry. The last character is currently unused. Permissions are indicated as follows:

- r** read permission is granted

- w write permission is granted
- a alter permission is granted
- indicated permission is *not* granted.

OWNER	(all)	Log-in name of the user who owns the facility entry.
GROUP	(all)	Group name of the group in which the owner of the facility entry belongs.
CREATOR	(a,c)	Log-in name of the user who created the facility entry.
CGROUP	(a,c)	Group name of the group in which the creator of the facility entry belongs.
CBYTES	(a,o)	Number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	Number of messages currently outstanding on the associated message queue.
QBYTES	(a,b)	Maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	Process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	Process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	Time the last message was sent to the associated queue.
RTIME	(a,t)	Time the last message was received from the associated queue.
CTIME	(a,t)	Time when the associated entry was created or changed.
NATTCH	(a,o)	Number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	Size of the associated shared memory segment.
CPID	(a,p)	Process ID of the creator of the shared memory entry.
LPID	(a,p)	Process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	Time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	Time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	Number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	Time the last semaphore operation was completed on the set associated with the semaphore entry.

CAUTIONS

The status of interprocess communication facilities can change so rapidly that the information produced by `ipcs` is only correct for the very moment that you query the system.

FILES

/etc/passwd usemames
/etc/group groupnames

RELATED INFORMATION

msgop(2), semop(2), shmop(2).

NAME

join – relational database operator

USAGE

join [*options*] *file1 file2*

DESCRIPTION

Join forms, on the standard output, a union between the two relations specified by the lines of *file1* and *file2*. If you use a dash (–) in place of *file1*, the standard input is used. *File1* and *file2* must be sorted in increasing ASCII collating sequence in the fields where they are to be joined (normally, the first in each line).

One line in the output for each pair of lines in *file1* and *file2* will have identical join fields. The output line normally consists of the common field, then the remainder of the line from *file1*, and finally, the remainder of the line from *file2*.

The default input field separators are blank, tab, or newline. Multiple separators usually count as one field separator, and leading separators are ignored. The default output field separator is a blank.

OPTIONS

Some of the options below use the argument *n*. This argument should be a 1 or 2 referring to either *file1* or *file2*, respectively.

- an* In addition to the normal output, produce a line for each line that cannot be paired in file *n*, where *n* is 1 or 2.
- e s* Replace empty output fields by string *s*.
- jn m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with one.
- o list* Let each output line comprise the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. Do not print the common field unless specifically requested.
- tc* Use character *c* as a separator (tab character) for both input and output. Every appearance of *c* in a line is significant.

EXAMPLE

To join the fourth field of both *file1* and *file2*, use this command:

```
# join -j1 4 -j2 4 file1 file2
```

CAUTIONS

With default field separation, the collating sequence is that of sort **-b**; with **-t**, the sequence is a plain sort.

The conventions used by **join** conflict with those used by **sort(1)**, **comm(1)**, **uniq(1)**, and **awk(1)**.

Numeric filenames may cause problems when you use the **-o** option just before listing filenames.

RELATED INFORMATION

awk(1), **comm(1)**, **sort(1)**, **uniq(1)**.

NAME

kill – terminate a process

USAGE

kill [*- signo*] *PID* ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes, unless an alternate signal is indicated. If no *- signo* (signal number) argument is supplied, all specified processes that do not catch or ignore the signal are stopped.

The Shell usually reports the process number of each asynchronous process beginning with an ampersand (&). If more than one process is started in a pipeline, however, the number of the last process in the pipeline is reported. Process numbers can also be found by using **ps**(1).

More detailed information about process termination is found in **kill**(2). Alternate signals that may be specified in *- signo* are described in **signal**(2).

EXAMPLES

To signal process 51 with an interrupt (2 is the signal generated):

```
# kill -2 51
```

To signal all processes in the process group to terminate (with the exception of the log-in Shell):

```
kill 0
```

CAUTIONS

Only the current owner of a process or the super-user may kill a process.

RELATED INFORMATION

ps(1), **sh**(1). **kill**(2), **signal**(2).

NAME

ld – link editor

USAGE

ld [**-r**] [**-d**] [**-o name**] *file* ... [**-l** [*x*]]

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries usually created by **ar**(1). It can also be used on libraries created by */com/lbr* (the DOMAIN librarian) and object modules produced by */com/bind* (the DOMAIN binder). When you specify several object *files*, **ld** combines them, producing an object module which can either be executed or become the input for a further **ld** run. If the module is to be used as input, you should supply the **-r** option to preserve the relocation bits. The output of **ld** is left on an *a.out* file.

Ld concatenates the argument routines in the order specified. If any argument is a library, **ld** searches it exactly once at the point encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The order of programs within libraries is unimportant.

OPTIONS

Except for **-l**, each of the options used should appear before filenames.

- l[x]** With an *x*, as in **-lx**, search */usr/lib/libx.a*, or */usr/local/lib/libx.a*, where *x* is a string. Searching is done when the library's name is encountered, so the placement of a **-l** is significant.
- r** Do not complain about undefined symbols.
- o name** Use *name* as the name of the **ld** output file, instead of *a.out*.
- e epsym** Set the default entry point address for the output file to be that of *epsym*.
- m** Produce a load mapfile of the linked object on the standard output.
- s** Strip line number entries and symbol table information from the output object file.
- L dir** Change the algorithm of searching for *libx.a* to look in *dir* before looking in */usr/lib* and */usr/local/lib*. This option is only effective if it precedes the **-l** option on the command line.
- M** Complain if external definitions do not match.
- VS num** Include a version num in the object.
- Wl,arg1,[arg2...]**
 Pass *argi* to the binder.

- T *systype*** Override the *systype* for which the input modules were compiled, and stamp the output module with the given *systype*. Use of this option changes the runtime semantics of the system calls and may cause the modules not to work. Its use is discouraged.
- f** Not supported.
- t** Not supported.
- u** Not supported.
- x** Not supported.
- V** Not supported.

FILES

<i>/usr/lib/lib?.a</i>	libraries
<i>a.out</i>	output file

RELATED INFORMATION

ar(1), *cc(1)*, *ar(4)*.

NAME

lex – generate programs for simple lexical tasks

USAGE

lex [-rctvn] [*file*] ...

DESCRIPTION

Lex generates programs for simple lexical analysis of text.

The input *files* contain strings and expressions to be searched for, and C text to be executed when strings are found. Lex treats multiple files as a single file. If no files are specified, it uses standard input.

Lex generates a C source program named *lex.yy.c* which, when loaded with the library, copies the input to the output, except where it encounters a specified string in the file being analyzed. The program then executes corresponding program text. The matching string remains in *yytext*, an external character array. Matching is done in the order that the strings appear in the file.

RULES

Strings may contain square brackets to indicate character classes, as in [*abx-z*] to indicate *a*, *b*, *x*, *y*, and *z*.

The operators *, +, and ? respectively signify any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class.

The period (.) is the class of all ASCII characters except newline.

Using parentheses for grouping and a vertical bar for alternation is allowed.

The notation *r*{*d,e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than a pipe (|), but lower precedence than the *, ?, and + characters, and concatenation.

The caret (^) at the beginning of an expression permits a successful match only immediately after a newline. A dollar sign (\$) at the end of an expression requires a trailing newline.

The slash (/) in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream.

An operator character may be used as an ordinary symbol if it is within double quotes (" ") or preceded by a backslash (\). Thus, [*a-zA-Z*]+ matches a string of letters.

Three subroutines defined as macros are expected. They are:

input() reads a character
unput(c) replaces a character read
output(c) places an output character

These subroutines are defined in terms of the standard streams, but you can override them. The program generated is named *yylex()*, and the library contains a *main()* that calls it. The action "REJECT" on the right side of the rule causes this match to be rejected and the next suitable match executed. The function *yymore()* accumulates additional characters into the same *yytext*. The function *yylless(p)* pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files *yyin* and *yyout* to read from and write to, defaulted to *stdin* and *stdout*, respectively.

Any line beginning with a blank is assumed to contain only C text and it is copied. If it precedes `,` it is copied into the external definition area of the *lex.yy.c* file. All rules should follow a double percent sign `(%)` as in *yacc(1)*. Lines preceding and beginning with a nonblank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with braces `{ }`. Note that braces do not imply parentheses; only string substitution is done.

The external names generated by *lex* all begin with the prefix *yy* or *YY*.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

p n number of positions is *n* (default 2000)
n n number of states is *n* (500)
t n number of parse tree nodes is *n* (1000)
a n number of transitions is *n* (3000)

Using one or more of the above automatically implies the `-v` option, unless the `-n` option is used.

OPTIONS

`-r` Specify RATFOR actions.
`-c` Indicate C actions (default).
`-t` Write the result of the lexical analysis on standard output, instead of in file *lex.yy.c* (default).
`-v` Provide a one-line summary of statistics of the generated analyzer.

-n Suppress printing of the one-line summary mentioned in the **-v** option (default).

EXAMPLE

```
D      [0-9]

if      printf("IF statement\n");
[a-z]+  printf("tag, value s\n",yytext);
0{D}+   printf("octal number s\n",yytext);
{D}+    printf("decimal number s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {      loop:
              while (input() != '*');
              switch (input())
              {
                case '/': break;
                case '*': unput('*');
                default: go to loop;
              }
            }
```

CAUTIONS

The **-r** option is not yet fully operational.

RELATED INFORMATION

yacc(1), malloc(3C).

NAME

line – read one line

USAGE

line

DESCRIPTION

Line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. Line is often used within Shell files to read from the user's terminal.

RELATED INFORMATION

sh(1), read(2).

NAME

lint – a C program checker

USAGE

lint [*options*] *file* ...

DESCRIPTION

Lint attempts to detect features of the C program files that are likely to be erroneous, nonportable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that **lint** currently detects are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, it checks the usage of functions to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but not returned.

Arguments with names ending in *.c* are considered C source files. Those with names ending in *.ln* are assumed to be the result of an earlier invocation of **lint** with either the **-c** or the **-o** option used. The *.ln* files are analogous to *.o* (object) files produced by the **cc(1)** command when given a *.c* file as input. **Lint** produces a warning message when it encounters files with other suffixes, but then it ignores these files.

Lint takes all the *.c*, *.ln*, and *llib-lx.ln* (specified by **-lx**) files and processes them in their command line order. By default, it appends the standard C **lint** library (*llib-lc.ln*) to the end of the list of files. However, if the **-p** option is used, **lint** appends the portable C **lint** library (*llib-port.ln*) instead. If the **-c** option is not used, the second pass of **lint** checks this list of files for mutual compatibility. When the **-c** option is used, the *.ln* and the *llib-lx.ln* files are ignored.

You may specify any number of **lint** options in any order, intermixed with filename arguments.

Lint produces its first output on a per-source-file basis. It collects and prints complaints regarding included files, after all source files have been processed. If the **-c** option is not used, **lint** collects information gathered from all input files and checks it for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, it prints the source filename followed by a question mark.

The behavior of the **-c** and the **-o** options allows for incremental use of **lint** on a set of C source files. Generally, one invokes **lint** once for each source file with the **-c** option. Each of these invocations produces a *.ln* file that corresponds to the *.c* file, and prints all messages about that source file only. After all the source files have been separately run through **lint**, invoking it once more (without the **-c** option), lists all the *.ln* files with the needed **-lx** options. This prints all the inter-file inconsistencies. This scheme works well with **make**, allowing **make** to detect only the source files that

have been modified since the last time the set of source files were run through **lint**.

Note that **lint** understands all DOMAIN C preprocessor directives, and that it also handles DOMAIN standard system calls properly.

OPTIONS

- a** Suppress complaints about assignments of long values to variables that are not long.
- b** Suppress complaints about *break* statements that cannot be reached. Programs produced by *lex* or *yacc* often result in many such complaints.
- h** Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.
- u** Suppress complaints about functions and external variables used and not defined, or defined and not used. Suitable for running **lint** on a subset of files of a larger program.
- v** Suppress complaints about unused arguments in functions.
- x** Do not report variables referred to by external declarations but never used.
- lx** Include the additional **lint** library *llib-lx.ln*. For example, you can include a **lint** version of the math library *llib-lm.ln* by inserting **-lm** on the command line. This argument does not suppress the default use of *llib-lc.ln*. These **lint** libraries must be in the assumed directory. This option can be used to reference local **lint** libraries and to develop multifile projects.
- n** Do not check compatibility against either the standard or the portable **lint** library.
- p** Attempt to check portability to other dialects (IBM and GCOS) of C. Truncate all nonexternal names to eight characters, and all external names to six characters and one case.
- c** Produce a *.ln* file for every *.c* file on the command line. These *.ln* files are the product of only the first pass of **lint**, and are not checked for interfunction compatibility.
- o lib** Create a **lint** library with the name *llib-lib.ln*. The **-c** option nullifies any use of the **-o** option. The **lint** library produced is the input given to the second pass of **lint**. The **-o** option simply causes this file to be saved in the named **lint** library. To produce a *llib-lib.ln* without extraneous messages, use **-x**. The **-v** option is useful if the source

file(s) for the lint library are just external interfaces (e.g., the way the file *llib-lc* is written). These option settings are also available by using certain conventional comments (see below).

Lint also recognizes the **-D**, **-U**, and **-I** options of **cpp(1)** and the **-g** and **-O** options of **cc(1)** as separate arguments. Although it ultimately ignores the **-g** and **-O** options, its recognition of these brings lint's behavior closer to that of the **cc** command.

The preprocessor symbol "lint" allows certain questionable code to be altered or removed for lint.

CONVENTIONAL COMMENTS

Certain conventional comments in the C source change the behavior of lint:

/*NOTREACHED*/	At appropriate points, stop comments about unreachable code. Typically placed just after calls to functions like exit(2) .
/*VARARGS<i>n</i>*/	Suppress the usual checking for variable numbers of arguments in the following function declaration. Check the data types of the first <i>n</i> arguments; interpret a missing <i>n</i> as zero.
/*ARGSUSED*/	Turn on the -v option for the next function.
/*LINTLIBRARY*/	At the beginning of a file, shut off complaints about unused functions and function arguments in this file. Equivalent to using the -v and -x options.

CAUTIONS

Lint does not understand functions that do not return, such as **exit(2)**.

In the case where an DOMAIN system call written in Pascal is called from C, lint sometimes complains about Pascal "out" variables that have not been initialized before use.

FILES

/usr/lib	location of lint libraries specified by the -lx option
/usr/lib/lint[12]	first and second passes
/usr/lib/llib-lc.ln	declarations for C library functions (binary format; source is in /usr/lib/llib-lc)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in /usr/lib/llib-port)
/usr/lib/llib-lm.ln	declarations for math library functions (binary format; source is in /usr/lib/llib-lm)
/usr/tmp/*lint*	temporaries

Note: All DOMAIN system calls have been added as declarations in the **lint** library */usr/lib/lldlib-lc.ln*. The system calls are found in the */sys/ins* files.

RELATED INFORMATION

cc(1), cpp(1), make(1).

NAME

ln – create a hard or soft link

USAGE

ln *name* [*target*]

ln -s *name target*

ln *name ... directory*

DESCRIPTION

ln creates both hard and soft links. A link is a directory entry that refers to a file. It is perfectly legal to have several links, in one or more directories, to the same file. Changes to a file are effective whether or not the file is referenced through a link.

A hard link is indistinguishable from the original directory entry. Hard links may not span file systems and may not refer to directories.

A soft (or symbolic) link contains a pathname. Symbolic links may span file systems and may refer to directories.

An **open(2)** operation on a link opens the referenced file. A **stat(2)** on a soft link is equivalent to a **stat** on the file that the link points to. Use **lstat(2)** to obtain information about the link itself. The **readlink(2)** call is useful for reading the contents of a soft link.

Given one or two arguments, **ln** creates a link to an existing file *name*. If *target* is given, the link has that name. The *target* argument may also be a directory in which to place the link. If *target* is not a directory, the link is placed in the current directory.

When the **-s** option is used, **ln** requires that a *target* be specified. If *target* exists, **ln -s** will fail. If only the directory is specified, the link is made to the last component of *name*.

Given more than two arguments, **ln** makes links to all the named files in the named directory. The links made will have the same name as the files being linked to.

OPTIONS

By default, **ln** generates a hard link.

-f Force creation of the link if permitted by access modes (hard links only).

-s Create soft (symbolic) links.

RELATED INFORMATION

cp(1)

mv(1)

rm(1)

link(2)

LN(1)

DOMAIN/IX SYS5

LN(1)

readlink(2)
lstat(2)
stat(2)

NAME

login – sign on

USAGE

login [*name* [*env-var* ...

DESCRIPTION

Login may be invoked by you as a command, or by the system when a connection is first established. If you invoke **login** as a command, it must replace the initial command interpreter. To do this, type

```
# exec login
```

or

```
# exec login name
```

at a shell prompt.

Login prompts for a name if none was supplied and, if appropriate, a password. The password is not echoed, so it will not appear on the transcript.

After a successful log-in, the message-of-the-day, if any, is printed and the user- and group-ID's are initialized. The working directory and command interpreter (shell) are then set as specified in */etc/passwd*. If no shell is specified in your password file entry, then the default command interpreter, */bin/sh*, is used.

If you change the shell field in your */etc/passwd* entry, you must also run */etc/crpasswd* to update the file */etc/passwd.map*. All other changes to */etc/passwd* (i.e., of home directory or password) must be made through the Registry. Normally, access to */etc/passwd* is restricted to *root*.

Login initializes the following environment variables:

```
HOME=your-login-directory  
PATH=:/bin:/usr/bin  
SHELL=last-field-of-passwd-entry  
MAIL=/usr/mail/your-login-name  
TZ=timezone-specification
```

The environment may be expanded or modified by supplying additional arguments to **login**, either at execution time or when your log-in name is requested. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as the following, where *n* is a number starting at zero and is incremented each time a new *variable* name is required:

Ln=xxx

Variables containing an equal sign (=) are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables *PATH* and *SHELL* cannot be changed. This prevents people who log into restricted Shell environments from spawning secondary Shells that are not restricted. Login understands simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

CAUTIONS

If you do not complete login successfully within a certain period of time (e.g., one minute), you may be silently disconnected.

FILES

<i>/usr/spool/mail/your-name</i>	mailbox for user <i>your-name</i>
<i>/etc/motd</i>	message-of-the-day
<i>/etc/passwd</i>	password file
<i>.profile</i>	log-in command file (Bourne Shell)

DIAGNOSTICS

“Login incorrect”: User name or the password cannot be matched.

“No shell,” “Cannot open password file,” or “no directory”: Consult system administrator.

RELATED INFORMATION

mail(1), sh(1), su(1), crpasswd(1M).

NAME

logname – get log-in name

USAGE

logname

DESCRIPTION

Logname returns the contents of the environment variable \$LOGNAME, which is set when you log into the system.

RELATED INFORMATION

env(1), login(1), logname(3X), environ(5).

NAME

lorder – find ordering relation for an object library

USAGE

lorder *file* ...

DESCRIPTION

Lorder uses as input one or more object or library archive *files* to find the ordering relation for an object library. The standard output lists pairs of object filenames, meaning that the first file of the pair refers to external identifiers defined in the second. You may want to process the output through **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**(1).

EXAMPLE

To build a new library from existing *.o* files, specify the following:

```
# ar cr library `lorder *.o | tsort`
```

CAUTIONS

The names of object files, in and out of libraries, must end with *.o*; otherwise, they are overlooked. Their global symbols and references are attributed to some other file.

FILES

**symref, *symdef*
temporary files

RELATED INFORMATION

ar(1), **ld**(1), **tsort**(1), **ar**(4).

NAME

lp, cancel – send/cancel requests to an **lp** line printer

USAGE

lp [-c] [-ddest] [-m] [-nnumber] [-ooption] [-s] [-ttitle] [-w] *file(s)*
cancel *ids* | *printers*

DESCRIPTION

Lp arranges for the named *file(s)* and associated information (collectively called a request) to be printed on a line printer. If no *file(s)* are mentioned, the standard input is assumed. A dash (-) used as a filename indicates the standard input; you may supply this character on the command line along with named *file(s)*. The order in which *file(s)* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. You may use this *id* to cancel the request, or you may use it to find the status (see **lpstat(1)**) of the request.

OPTIONS

The following options to **lp** may appear in any order and may be intermixed with filenames:

- c** Make copies of the *file(s)* to be printed immediately when **lp** is invoked. Normally, *file(s)* will not be copied, but will be linked. If the **-c** option is not given, then you should be careful not to remove any of the *file(s)* before the request has been printed completely. Without the **-c** option, any changes made to the filename(s) after the request is made, but before it is printed, will be reflected in the printed output.
- ddest** Choose *dest* as the printer or class of printers where printing will take place. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailable, file space limitation, etc.), requests for specific destinations may not be accepted (see **accept(1M)** and **lpstat(1)**). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see **lpstat(1)**).
- m** Send mail after the *file(s)* have been printed (see **mail(1)**). By default, no mail is sent.
- nnumber** Print *number* copies (default = 1) of the output.
- ooption** Specify a printer-dependent or class-dependent *option*. Several such *options* may be collected by specifying the **-o** keyletter more than once.

For more information about what is valid for *options*, see **Models in lpadmin(1M)**.

- s** Suppress messages from **lp(1)** such as "request id is ...".
- ttitle** Print *title* on the banner page of the output.
- w** Write a message on the user's terminal after the *file(s)* have been printed. If the user is not logged in, then send mail.

Cancel cancels line printer requests that were made by the **lp(1)** command. The command line arguments may be either request *ids* (as returned by **lp(1)**) or *printer* names. For a complete list of *printer* names, use **lpstat(1)**. Specifying a request *id* cancels the associated request, even if it is currently printing. Specifying a *printer* cancels the request that is currently printing on that *printer*. In either case, the cancellation of a request that is currently printing frees the *printer* to print its next available request.

FILES

*/usr/spool/lp/** spooling directories

RELATED INFORMATION

enable(1)
lpstat(1)
mail(1)
accept(1M)
lpadmin(1M)
lpsched(1M)

NAME

lpstat – print **lp** status information

USAGE

lpstat [*options*] [*ids*]

DESCRIPTION

lpstat prints information about the current status of the **lp** line printer system.

If no *options* are specified, **lpstat** prints the status of all requests that you have made to **lp(1)**. Any arguments that are not *options* are assumed to be request *ids* (as returned by **lp**). **lpstat** prints the status of such requests.

OPTIONS

Options may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

Omitting the *list* following a keyletter causes all information relevant to the keyletter to be printed.

- a**[*list*] Print acceptance status (with respect to **lp**) of destinations for requests. *List* is a list of intermixed printer names and class names.
- c**[*list*] Print class names and their members. *List* is a list of class names.
- d** Print the system default destination for **lp**.
- o**[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.
- p**[*list*] Print the status of printers. *List* is a list of printer names.
- r** Print the status of the **lp** request scheduler
- s** Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.
- t** Print all status information.
- u**[*list*] Print status of output requests for users. *List* is a list of log-in names.
- v**[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

*/usr/spool/lp/**

spooling directories

RELATED INFORMATION

enable(1)

lp(1)

NAME

ls – list contents of directory

USAGE

ls [-RSadCxmlnogrtucpFbqisf] [*names*]

DESCRIPTION

For each directory argument, ls lists the contents of the directory. For each file argument, ls repeats its name and any other information requested. By default, it sorts the output alphabetically. If you specify no argument, ls lists the current directory. If you give several arguments, ls first sorts the arguments appropriately, but prints file arguments before directories and their contents.

Ls can produce lists in three major formats. By default, it lists one entry per line. It can also generate a multi-column format, as well as stream output format in which files are listed across the page, separated by commas.

OPTIONS

- R** Recursively list subdirectories encountered. This option will follow soft links unless the **-S** option is also used.
- S** Shows link text rather than the object to which the link has been made.
- a** List all entries. Usually, entries whose names begin with a period (.) are not listed.
- d** If an argument is a directory, list only its name (not its contents). Often used with **-l** to get the status of a directory.
- C** Produce multi-column output with entries sorted down the columns.
- x** Produce multi-column output with entries sorted across rather than down the page.
- m** Produce stream output format.
- l** List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, the size field contains the major and minor device numbers, rather than a size. The mode printed under the **-l** option consists of 10 characters, interpreted as follows:

The first character is:

- d** directory
- b** block special file
- c** character special file

XX is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires *XX* to be different from the suffix of the archive member. Thus, you cannot have *lib(file.o)* depend upon *file.o* explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C-type source:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
      @echo lib is now up-to-date
.c.a:
      $(CC) -c $(CFLAGS) $<
      ar rv $@ $*.o
      rm -f $*.o
```

In fact, the *c.a* rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
      $(CC) -c $(CFLAGS) $(?:.o=.c)
      ar rv lib $?
      rm $? @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The *\$?* list is defined to be the set of object filenames (inside *lib*) whose C source files are outdated. The substitution mode translates the *.o* to *.c*. Note that the *.c.a:* rule, which would have created each object file one by one, has been disabled. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

CAUTIONS

Some commands return non-zero status inappropriately; use *-i* to overcome this.

Filenames with the equal sign (=), colon (:), or at sign (@) will not work.

Commands directly executed by the Shell, notably *cd(1)*, are ineffectual across newlines in *make*.

The syntax (*lib(file1.o file2.o file3.o)*) is illegal. You cannot build *lib(file.o)* from *file.o*.

The macro *\$(a:.o=.c~)* does not work.

MAKE (1)

DOMAIN/IX SYS5

MAKE (1)

FILES

[Mm]akefile
s.[Mm]akefile

RELATED INFORMATION

cc(1), cd(1), lex(1), sh(1), yacc(1), printf(3S), sccsfile(4).

NAME

man – print entries in this manual

USAGE

man [*options*] [*section*] *title(s)*

DESCRIPTION

Man locates and prints the manual page(s) specified by the *title* argument. If you also specify a *section*, **man** looks for the *title* only in the *section* indicated (otherwise, it searches the whole manual). You must enter the *title* in lowercase, and the *section* number may not have a letter suffix.

Man examines the environment variable \$TERM and attempts to select options that adapt the output to the terminal being used. Refer to **environ(5)** for more information. The **-Tterm** option overrides the value of \$TERM.

Section may be changed before each *title*.

OPTIONS

- Tterm** Print the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type **help term2**. Also refer to **term(5)** for further details. The default value of *term* is 450.
- w** Print on the standard output only the *pathnames* of the entries, relative to */usr/catman*, or to the current directory for **-d** option.
- d** Search the current directory rather than */usr/catman*. This option requires that you specify the full filename (e.g., *cu.1c*, rather than just *cu*).
- c** Invoke **col(1)**. This is done automatically, unless *term* is one of the following: 300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620, and X.

CAUTIONS

Man prints manual entries that were formatted by **nroff(1)** when the UNIX system was installed. Entries are originally formatted with terminal type 37, and are printed using the correct terminal filters as derived from the **-Tterm** and \$TERM settings. Certain typesetting or other nonstandard printing of manual entries may require installation of the UNIX System V Documenter's Workbench Software (not currently supported by Apollo). Check the *DOMAIN/IX Text Processing Guide* to see what text tools are generally available for use now.

EXAMPLE

To reproduce this page on the terminal, as well as any other entries named **man** that may exist in other sections of the manual, use:

```
# man man
```

FILES

*/usr/catman/?_man/man[1-8]/** preformatted manual entries; system and command reference directories

RELATED INFORMATION

col(1)

term(5)

NAME

mesg – permit or deny messages

USAGE

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** revokes nonuser write permission on your terminal. Mesg with argument **y** reinstates permission. Mesg without an argument reports your current decision (y or n) regarding the receipt of messages, but does not change it.

Note: This command is a no-op in DOMAIN/IX, thus allowing the shell script that calls it to run.

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir – make a directory

USAGE

mkdir *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (all access permissions granted). It automatically makes standard entries of “.” for the directory itself, and “..” for its parent.

CAUTIONS

Mkdir requires write permission in the parent directory.

Umask(1) may alter specified directories normally created with mode 777, causing them to become mode 755.

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

RELATED INFORMATION

sh(1), **rm(1)**, **umask(1)**.

NAME

mm – print out documents formatted with the MM macros

USAGE

mm [*options*] [*files*]

DESCRIPTION

Mm can be used to type out documents using **nroff(1)** and the **mm(5)** text-formatting macro package. It has options to specify preprocessing by **tbl(1)** and/or **neqn(1)** and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for **nroff(1)** and the **mm** macros are generated, depending on the options selected.

The valid *options* are given below. Any other arguments or flags (e.g., **-rC3**) are passed to **nroff(1)** or to the **mm** macro package as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, **mm** prints a list of its options.

Mm reads the standard input when a dash (–) is specified instead of any filenames. This option allows **mm** to be used as a filter, e.g.:

```
cat dws | mm –
```

OPTIONS

- | | |
|---------------|--|
| -Tterm | Specify the type of output terminal. For a list of recognized values for <i>term</i> , type help term2 . If this option is <i>not</i> used, mm uses the value of the shell variable \$TERM from the environment as the value of <i>term</i> if \$TERM is set. Refer to profile(5) and environ(5) for more information. Otherwise, mm uses 450 as the value of <i>term</i> . If several terminal types are specified, the last one takes precedence. |
| -12 | Produce the document in 12 pitch. This option may be used when \$TERM is set to one of the following: 300, 300s, 450, and 1620. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.) |
| -c | Invoke col(1) . This is done automatically, unless <i>term</i> is one of the following: 300, 300s, 450, 37, 4000A, 382, 4014, tek, 1620, and X. |
| -e | Invoke neqn(1) . |
| -t | Invoke tbl(1) . |
| -E | Invokes the -e option of nroff(1) . |
| -y | Use the non-compacted version of the macros. |

EXAMPLES

Assuming that the shell variable \$TERM is set in the environment to 450, the two command lines below are equivalent:

```
# mm -t -rC3 -12 ghh*  
# tbl ghh* | nroff -cm
```

CAUTIONS

Mentioning other files along with the dash (-) that denotes standard input produces unfavorable results.

Mm invokes **nroff** with the **-h** flag. With this flag, **nroff** assumes that the terminal has tabs set every eight character positions.

Use the **-olist** option of **nroff** to specify ranges of pages to be output. Note, however, that invoking **mm** with one or more of the **-e**, **-t**, and **-** options, *together* with the **-olist** option of **nroff**, may cause a harmless “broken pipe” diagnostic if the last page of the document is not specified in *list*.

If you use the **-s** option of **nroff** to stop between pages of output, use line-feed rather than return or newline to restart the output. The **-s** option of **nroff** does not operate properly with the **-c** option of **mm**, or if **mm** automatically invokes **col**.

If you do not correctly specify the type of terminal on which the output of **mm** is to be printed, you will probably get a garbage message back. However, if you are redirecting output into a file, use the **-T37** option, and then use the appropriate terminal filter when you actually print that file.

DIAGNOSTICS

“mm: no input file” None of the arguments is a readable file and **mm** is not used as a filter

RELATED INFORMATION

col(1), **env(1)**, **eqn(1)**, **mmt(1)**, **nroff(1)**, **tbl(1)**.

DOMAIN/IX Text Processing Guide.

NAME

mmt, **mvt** – typeset documents, viewgraphs, and slides

USAGE

mmt [*options*] [*files*]

DESCRIPTION

Although very similar to **mm**(1), these two commands typeset their input via **troff**(1), as opposed to formatting it via **nroff**(1). **Mmt** uses the MM macro package for its operations. **Mvt** uses the Macro Package for View Graphs and Slides. Preprocessing through **tbl**(1) and **eqn**(1) is available for both. The proper pipelines and the required arguments and flags for **troff**(1) and the appropriate macro packages are generated, depending on the options selected.

Arguments or flags other than those given below are passed to **troff**(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* argument. If no arguments are given, **mmt** prints a list of its options.

OPTIONS

- e** Invoke **eqn**(1) and cause it to read the */usr/pub/eqnchar* file. See *eqnchar*(5) for details concerning this file.
- t** Invoke **tbl**(1).
- Tdest** Create output for **troff**(1) device *dest*.
- Di10** Direct the output to the local Imagen Imprint-10 laser printer.
- a** Invoke the **-a** option of **troff**(1).
- y** Use the noncompacted version of the macros. This is the default.
- z** Invoke no output filter to process or redirect the output of **troff**(1).

CAUTIONS

If you specify a simple dash (**-**), **-e** option, and/or **-t** option along with the **-olist** option of **troff**(1), a harmless “broken pipe” diagnostic may result. This usually happens under these conditions if the last page of the document is not specified in *list*.

DIAGNOSTICS

“m[mv]t: no input file” if none of the arguments is a readable file and the command is not used as a filter.

Mvt is just a link to **mmt**.

RELATED INFORMATION

env(1), eqn(1), mm(1), nroff(1), tbl(1), troff(1), environ(5), mm(5), mv(5).

DOMAIN/IX Text Processing Guide.

NAME

mv – move files

USAGE

mv [**-f**] *file1* [*file2* ...] *target*

DESCRIPTION

Mv moves *file(s)* to a specified *target*. Under no circumstances can any of the *files* being manipulated be the same as the *target*, so take care when using shell metacharacters. If *target* is a directory, then the *file(s)* will be moved to that directory. If *target* is a file, its old contents are replaced by the contents of *file*.

If **mv** determines that the mode of *target* forbids writing, it prints the mode, asks for a response, and reads the standard input for one line. If that line begins with **y**, the operation occurs if it is permissible; if not, **mv** exits. If the standard input is not a terminal, or if the **-f** (force) option is used, the **mv** will be performed, if permitted, with no questions asked.

If *file1* is a directory, the directory rename occurs only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file. If *target* is not a file, **mv** creates a new file with the same mode as *file1*. The owner and group of *target* are those of the user. If *target* is a file, moving a file into *target* does not change *target*'s mode, owner, or group. **Cp** sets the last modification time of *target*, (and last access time, if *target* did not exist). If *target* is a link to a file, all links remain and the file is changed.

OPTIONS

-f Force the operation if it is permissible. Do not ask for confirmation.

RELATED INFORMATION

chmod(1)

cp(1)

cpio(1)

rm(1)

NAME

newform – change the format of a text file

USAGE

newform [-s] [-itabspec] [-otabspec] [-bn] [-en] [-pn] [-an] [-f]
[-cchar] [-ln] [files]

DESCRIPTION

Newform reads lines from the named *files* (or the standard input if no input file is named) and reproduces the lines on the standard output. It reformats lines according to command line options in effect.

Except for -s, options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Newform processes options in the order specified. This means that option sequences like -e 15 -l 60 yield results different from -l 60 -e 15. Options are applied to all *files* on the command line.

OPTIONS

- itabspec Input tab specification: expand tabs to spaces, according to given tab specifications. *Tabspec* recognizes all tab specification forms described in *tabs(1)*. In addition, *tabspec* may be a dash (-), signifying that the tab specification is to be found in the first line read from the standard input. If no *tabspec* is given, *tabspec* defaults to -8. A *tabspec* of -0 expects no tabs; if any are found, they are treated as -1.
- otabspec Output tab specification: replace spaces by tabs, according to given tab specifications. The tab specifications are the same as for -itabspec. If no *tabspec* is given, *tabspec* defaults to -8. A *tabspec* of -0 means that no spaces are converted to tabs on output.
- ln Set the effective line length to *n* characters. If *n* is not entered, -l defaults to 72. The default line length without the -l option is 80 characters. Note that tabs and backspaces are considered to be one character (use -i to expand tabs to spaces).
- bn Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see -ln). If no *n* is specified, truncate the number of characters necessary to obtain the effective line length.
- en Same as -bn except that characters are truncated from the end of the line.
- cchar Change the prefix/append character to *char*. By default, this character is a space.
- pn Prefix *n* characters (see -ck) to the beginning of a line when the line length is less than the effective line length. The default is to prefix the number of characters necessary to obtain the effective line length.

- an** Same as **-pn** except characters are appended to the end of a line.
- f** Write the tab specification format line on the standard output before any other lines are output. The tab specification format line that is printed corresponds to the format specified in the *last* **-o** option. If no **-o** option is specified, the line that is printed contains the default specification of **-8**.
- s** Shear off leading characters on each line up to the first tab and place up to eight of the sheared characters at the end of the line. If more than eight characters (not counting the first tab) are sheared, replace the eighth character by an asterisk (*) and discard any characters to the right of it. The first tab is always discarded.
- An error message and program exit occurs if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

EXAMPLES

To delete the sequence numbers from a COBOL program file, use the **newform** command as follows:

```
newform -l1 -b7 filename
```

(The **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated).

To convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, specify this::

```
newform -s -i -l -a -e filename
```

CAUTIONS

Newform normally only keeps track of physical characters. However, if you use the **-i** and **-o** options, **newform** keeps track of backspaces so that tabs can be lined up in the appropriate logical columns.

Newform does not prompt you if a *tabspec* is to be read from the standard input.

If the **-f** option is used, and the last **-o** option specified was **-o- -**, and was preceded by either a **-o- -** or a **-i- -**, the tab specification format line will be incorrect.

DIAGNOSTICS

All diagnostics are fatal.

“usage: ...”	bad option used
“not -s format”	a line is missing a tab
“can’t open file”	self-explanatory
“internal line too long”	a line exceeds 512 characters after being expanded in the internal work buffer
“tabspec in error”	a tab specification is incorrectly formatted, or specified tab stops are not ascending
“tabspec indirection illegal”	a <i>tabspec</i> read from a file (or standard input) may not contain a <i>tabspec</i> referencing another file (or standard input)

EXIT CODES

- 0 – normal execution
- 1 – for any error

RELATED INFORMATION

csplit(1), tabs(1).

NAME

newgrp – log in to a new group

USAGE

newgrp [-] [*group*]

DESCRIPTION

The **newgrp** command changes your group identification. Although you remain logged in during the process, and your current directory is unchanged, **newgrp** sets new real and effective group IDs. The shell then performs calculations of access permissions to files with respect to these new IDs. You are always given a new shell to replace the current shell, regardless of whether **newgrp** terminates successfully or due to an error condition (e.g., unknown group).

Exported variables retain their values after you invoke **newgrp**. All unexported variables, however, are either reset to their default value or set to null. Unless you or the system itself exports system variables (e.g., PS1, PS2, PATH, MAIL, HOME), they are reset to default values. For example, suppose you have a primary prompt string (PS1) other than the default, a pound sign (#), and you have not exported PS1. After invoking **newgrp**, successfully or not, your PS1 variable is set to the default prompt string, the pound sign (#). Use the shell command, **export**, to export variables so that they retain their assigned value when invoking new shells. See **sh(1)** for more information.

With no arguments, **newgrp** changes the group identification back to the group specified in the your password file entry. If the first argument to **newgrp** is a dash (-), the environment changes to one that you would normally expect if you logged in again.

The **newgrp** command lets you change to any group of which you are a member. The */etc/group* file contains a list of all groups and the group's members. This information is compiled by **crpasswd(1M)** from the registry files. You are a member of all groups for which you have an account. For example, if you have the following three registry accounts,

```
user1.project1.org
user1.project2.org
user1.project3.org
```

you are listed three times in the */etc/group* file.

The */etc/passwd* file contains your default group. Even though this may not appear in the */etc/group* file, this group is always available as an option to the **newgrp** command.

CAUTIONS

DOMAIN/IX does not allow group passwords. Group passwords encourage poor security practices.

FILES

<i>/etc/group</i>	system's group file
<i>/etc/passwd</i>	system's password file

RELATION INFORMATION

- login(1)
- sh(1)
- group(4)
- passwd(4)
- environ(5)

NAME

news – print news items

USAGE

news [**-a**] [**-n**] [**-s**] [*items*]

DESCRIPTION

News keeps you informed of current events. By convention, these events are described by files in the directory */usr/news*.

By using the *items* argument, you may indicate specific news items to be printed. If you do not specify any arguments, **news** prints the contents of all current files in */usr/news*, the most recent one first, with each preceded by an appropriate header. News stores the “currency” time as the modification date of a file named *.news_time* in your home directory. The identity of this directory is determined by the environment variable \$HOME. Only files more recent than this currency time are considered “current.”

If you type a *delete* during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

OPTIONS

- a** Print all items, regardless of currency. Do not change the stored time.
- n** Report the names of the current items without printing their contents, and without changing the stored time.
- s** Report how many current items exist, without printing their names or contents, and without changing the stored time. Useful to include in your *.profile* file, or in the system’s */etc/profile*.

FILES

/etc/profile
*/usr/news/**
\$HOME/.news_time

RELATED INFORMATION

environ(5).

NAME

nice – run a command at low priority

USAGE

nice [*-increment*] *command* [*arguments*]

DESCRIPTION

Nice executes a specified *command* with a lower CPU scheduling priority. You may specify an *increment* argument in the range of -19 through +19 (the default is +10). A specified increment larger than 19 will still be equivalent to 19. To run commands with a higher than normal priority, use a negative increment (e.g., **nice - -10**). To run them at a lower priority, use a positive increment. The higher the number in positive terms, the lower the priority will be for running a command.

DIAGNOSTICS

Nice returns the exit status of the subject command.

RELATED INFORMATION

nohup(1), **nice**(2).

NAME

nl – line numbering filter

USAGE

nl [**-h***type*] [**-b***type*] [**-f***type*] [**-v***start#*] [**-i***incr*] [**-p**]
 [**-l***num*] [**-s***sep*] [**-w***width*] [**-n***format*] [**-d***delim*] *file*

DESCRIPTION

Nl reads lines from the named *file* (or the standard input if no *file* is named) and reproduces the lines on the standard output. Lines are numbered on the left according to the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line-numbering options are independently available for the header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

<i>Line contents</i>	<i>Start of</i>
\\	header
\	body
	footer

Unless optioned otherwise, **nl** assumes the text being read is in a single logical page body. Command options may appear in any order and may be intermingled with an optional filename. Only one file at a time may be named.

OPTIONS

- b***type* Specify which logical page body lines are to be numbered. Recognized *types* are as follows: *a*, number all lines; *t*, number lines with printable text only; *n*, number no lines; *p string*, number only lines containing the regular expression specified in *string*. The default *type* for the logical page body is *t* (text lines numbered).
- h***type* Same as **-b***type* except for the header. The default *type* for the logical page header is *n* (no lines numbered).
- f***type* Same as **-b***type* except for the footer. The default for the logical page footer is *n* (no lines numbered).
- p** Do not restart numbering at logical page delimiters.

- vstart#** Use *start#* as the initial value used for numbering logical page lines. The default is one.
- incr** Use *incr* as the increment value for numbering logical page lines. The default is one.
- ssep** Use *sep* as the character(s) to separate the line number and the corresponding text line. The default separator is a tab.
- wwidth** Use *width* as the number of characters to be used for the line number. The default *width* is six.
- nformat** Use *format* as the line-numbering format. The recognized values are: *ln*, left justified, leading zeros suppressed; *rn*, right justified, leading zeros suppressed; *rz*, right justified, leading zeros kept. The default *format* is *rn* (right justified).
- lnum** Use *num* as the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). The default is one.
- ddelim** Change the delimiter characters specifying the start of a logical page section from the default characters (**\:**) to *delim*, which should be two user-specified characters. If only one character is entered, the second character remains the default character (**:**). No space should appear between the **-d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

To number *file1* starting at line number 10 with an increment of 10, use the following command:

```
# nl -v10 -i10 -d!+ file1
```

The logical page delimiters are **!+**.

RELATED INFORMATION

pr(1).

NAME

nm – print name list

USAGE

nm [**-gnopru**] [*file* ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If you fail to supply the name of a *file*, **nm** lists the symbols in *a.out*.

Each symbol name is preceded by its value (blanks if undefined) and one of the following letters: U (undefined), A (absolute), T (text segment symbol), D (data segment symbol), B (bss segment symbol), C (common symbol), f (file name), or G (external global text symbol). If the symbol is local (non-external), the type letter is in lower case. **Nm** sorts the output alphabetically.

OPTIONS

- g** Do not print global (external) symbols.
- n** Sort numerically rather than alphabetically.
- o** Prepend filename to each output line.
- p** Don't sort; print in symbol-table order.
- r** Sort in reverse order.
- u** Print only symbols that are undefined (runtime libraries are checked).

RELATED INFORMATION

ar(1), **ar**(4).

NAME

nohup – run a command immune to hangups and quits

USAGE

nohup *command* [*arguments*]

DESCRIPTION

Nohup executes a specified *command* with hangups and quits ignored. If you do not redirect output, both standard output and standard error are sent to *nohup.out*. If *nohup.out* is not writable in the current directory, output is redirected to *\$HOME/nohup.out*.

EXAMPLES

It is frequently desirable to apply **nohup** to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a Shell procedure, and then issuing the following:

```
nohup sh file
```

The **nohup** then applies to everything in *file*. If the Shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. See *sh(1)* for general information about the Shell.

Add an ampersand (&) and the contents of *file* are run in the background with interrupts also ignored, as shown here:

```
nohup file &
```

The contents of *file* could be the following:

```
tbl ofile | eqn | nroff > nfile
```

CAUTIONS

If you specify **nohup** *command1*; *command2*, **nohup** applies only to *command1*; **nohup** (*command1*; *command2*) is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
# nohup cpio -o <list >/dev/rmt/lm&
```


On the other hand, this command puts the error messages into the file *errors*:

```
# nohup cpio -o <list >/dev/rmt/lm
```

RELATED INFORMATION

chmod(1), nice(1), sh(1), signal(2).

NAME

nroff – format text

USAGE

nroff [*options*] [*files*]

DESCRIPTION

Nroff formats text found in *files* (standard input by default) for printing on line printers, letter quality printers, or similar devices. An argument consisting of a dash (-) is interpreted as a filename corresponding to the standard input.

The options below may appear in any order, but must appear before the *files*.

Note: DOMAIN/IX **nroff** is the same as the **nroff** shipped with Unix System III.

OPTIONS

- olist** Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N-M* means pages *N* through *M*; an initial *-N* means from the beginning to page *N*; and a final *N-* means from *N* to the end.
- nN** Number first generated page *N*.
- raN** Set register *a* (which must have a one-character name) to *N*.
- i** Read standard input after *files* are exhausted.
- q** Invoke the simultaneous input/output mode of the **.rd** request.
- z** Print only messages generated by **.tm** (terminal message) requests.
- mname** Prepend to the input *files* the noncompacted (ASCII text) macro file */usr/lib/tmac/tmac.name*.
- cname** Prepend to the input *files* the compacted macro files. These files are named */usr/lib/macros/cmp.[nt].[dt].name* and */usr/lib/macros/ucmp.[nt].name*.
- kname** Compact the macros used in this invocation, placing the output in *[dt].name* files in the current directory.
- sN** Stop every *N* pages (the default is one) to allow paper loading or changing. Resume after receiving a linefeed or newline. Note that newlines do not work in pipelines, e.g., with **mm(1)**. This option does not work if the output of **nroff** is piped through **col(1)**. When **nroff** halts between pages, an ASCII BEL is sent to the terminal.
- Tname** Prepare output for a specified terminal. Known *names* are:
 - 37 TELETYPE Model 37 terminal (or equivalent)

- tn300* GE TermiNet 300 (or any terminal without half-line capability)
- 300s* DASI 300s
- 300* DASI 300
- 450* DASI 450
- 382* DTC-382
- 4000A* Trendata 4000A
- 832* Anderson Jacobson 832
- X* generic EBCDIC printer
- e** Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.
- h** Use output tabs during horizontal spacing to speed output and reduce output character count. The tab settings are assumed to be at every eight nominal character widths.
- un** Set the boldface factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

CAUTIONS

Nroff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that **nroff** generates may be off by one day from your idea of what the date is.

Using **nroff** with the **-olist** option inside a pipeline, may cause a harmless “broken pipe” diagnostic if the last page of the document is not specified in *list*.

FILES

<i>/usr/lib/suftab</i>	suffix hyphenation tables
<i>/tmp/ta\$#</i>	temporary file
<i>/usr/lib/tmac/tmac.*</i>	standard macro files and pointers
<i>/usr/lib/macros/*</i>	standard macro files
<i>/usr/lib/term/*</i>	terminal driving tables for nroff

RELATED INFORMATION

col(1), **cw(1)**, **eqn(1)**, **mm(1)**, **mmt(1)**, **tbl(1)**, **troff(1)**.

NAME

od – octal dump

USAGE

od [**-bcdosx**] [*file*] [[**+**] *offset* [**.**] [**b**]]

DESCRIPTION

Od dumps *file* in one or more selected formats. If none of the options below are specified, *file* is interpreted in octal by default.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If a period (.) is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by a plus sign (+).

Dumping continues until end-of-file.

OPTIONS

- b** Interpret bytes in octal.
- c** Interpret bytes in ASCII. Certain nongraphic characters appear as C escapes: null=**\0**, backspace=**\b**, formfeed=**\f**, newline=**\n**, return=**\r**, tab=**\t**; others appear as three-digit octal numbers.
- d** Interpret words in unsigned decimal.
- o** Interpret words in octal.
- s** Interpret 16-bit words in signed decimal.
- x** Interpret words in hexadecimal.

NAME

pack, pcat, unpack – compress and expand files

USAGE

pack [-] [-f] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible and useful, it replaces each input file *name* by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*.

If **pack** is successful, it removes *name*. Packed files can be restored to their original form using **unpack** or **pcat**.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75 percent of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90 percent of the original size. **Pack** returns a value equaling the number of files not compressed.

No packing occurs if one or more of the following conditions exists:

- the file appears to be already packed
- the filename has more than 12 characters
- the file has links
- the file is a directory
- the file cannot be opened
- no disk storage blocks will be saved by packing
- a file called *name.z* already exists
- the .z file cannot be created
- an I/O error occurred during processing.

The last segment of the filename must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what **cat(1)** does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name.z* use:

pcat name.z
or just:
pcat name

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

pcat name > nnn

Pcat returns the number of files it was unable to unpack, but will fail if one of the following conditions exist:

- the filename (exclusive of the *.z*) has more than 12 characters
- the file cannot be opened
- the file does not appear to be the output of **pack**.

Unpack expands files created by **pack**. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in *.z*). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the *.z* suffix stripped from its name. It also has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. It will fail for the same reasons as those listed for **pcat**, as well as for the following additional reasons:

- a file with the “unpacked” name already exists
- the unpacked file cannot be created.

OPTIONS

(Note that these options are only for use with **pack**.)

- Set an internal flag that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* cause the internal flag to be set and reset.
- f Force packing of *name*. Useful for causing an entire directory to be packed even if some of the files will not benefit.

PACK (1)

DOMAIN/IX SYS5

PACK (1)

RELATED INFORMATION
cat(1).

NAME

passwd – change log-in password

USAGE

passwd *name*

DESCRIPTION

This command changes (or installs) a password associated with the log-in *name*.

The program prompts for the old password (if any) and then for the new one (twice). You must supply these. New passwords should be at least four characters long, if you want to use a mixture of uppercase and lowercase. They should be at least six characters long if you choose to format them in a single case. Only the first eight characters of the password are significant to the program.

Only the owner of the name or the super-user may change a password; the owner must prove knowledge of the old password. Only the super-user can create a null password.

On DOMAIN systems, passwords are not stored in */etc/passwd*; instead, the site's local registry stores them.

RELATED INFORMATION

login(1)
crypt(3C)
passwd(4)
crpasswd(1M)

NAME

paste – merge same lines of several files or subsequent lines of one file

USAGE

paste [-ds] [*list*] *file1 file2 ...*

DESCRIPTION

If you specify no options, or simply the **-d** option, paste concatenates corresponding lines of the given input files (*file1*, *file2*, etc.). It treats each file as one or more columns of a table and pastes them together horizontally (also known as parallel merging). You may specify a dash (–) in place of a filename, causing paste to read from the standard input (note that there is no prompting for this).

Paste is the counterpart of cat(1), which concatenates vertically (i.e., one file after the other).

If you specify the **-s** option, paste replaces the function of an older command with the same name by combining subsequent lines of the input file (also known as serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if a dash is used in place of a filename.

OPTIONS

- | | |
|-------------|---|
| -d | Do not replace the newline characters of each but the last file (or last line) by a <i>tab</i> character. Allow replacement of the <i>tab</i> character by one or more alternate characters (see below). |
| <i>list</i> | Replace the default <i>tab</i> as the line concatenation character with the one or more characters immediately following -d . When the list is exhausted, it is reused. In parallel merging (i.e., no -s option), the lines from the last file are always terminated with a newline character, not from the <i>list</i> . The list may contain these special escape sequences: \n (newline), \t (tab), \\ (backslash), and \0 (empty string, not a null character). Quotation marks may be necessary, if characters have special meaning to the Shell (e.g., to get one backslash, use -d “\”). |
| -s | Merge subsequent lines rather than one from each input file. Use <i>tab</i> for concatenation, unless a <i>list</i> is specified with the -d option. Regardless of the <i>list</i> , the very last character of the file is forced to be a newline. |

EXAMPLES

To list the current directory in three columns, use the following command:

```
ls | paste - - -
```

To combine pairs of lines into lines in *file*, use this command:

```
paste -s -d "\t\n" file
```

DIAGNOSTICS

“line too long”

Output lines are restricted to 511 characters.

“too many files”

Except for the `-s` option, no more than 12 input files may be specified.

RELATED INFORMATION

`cut(1)`, `grep(1)`, `pr(1)`.

NAME

pg – file perusal filter for soft-copy terminals

USAGE

pg [*- number*] [*-pstring*] [*-cefns*] [*+linenumber*] [*+/pattern/*] [*files ...*]

DESCRIPTION

The **pg** command is a filter that lets you examine *files* one screenful at a time on a soft-copy terminal. A dash (–) and/or any null arguments supplied in place of *files* causes **pg** to read from the standard input. Each screenful is followed by a prompt. If you type a carriage return, **pg** displays another page; other possibilities are enumerated below.

This command differs from other paginators in that it lets you back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, **pg** scans the **terminfo**(4) database for the terminal type specified by the **TERM** environment variable. If **TERM** is undefined, **pg** assumes the terminal type *dumb*.

OPTIONS

- | | |
|------------------|--|
| -number | Specify window size as <i>number</i> lines. (On a terminal containing 24 lines, the default window size is 23). |
| -p string | Use <i>string</i> as the prompt. If the prompt string contains a “d”, the first occurrence of “d” in the prompt is replaced by the current page number when the prompt is issued. The default prompt string is a colon (:). |
| -c | Home the cursor and clear the screen before displaying each page. This option is ignored if <i>clear_screen</i> is not defined for this terminal type in the terminfo (4) database. |
| -e | Do not pause at the end of each file. |
| -f | Inhibit line-splitting tendencies. Normally, pg splits lines longer than the screen width, but some character sequences in the text being displayed (e.g., escape sequences for underlining) generate undesirable results unless this option is used. |
| -n | Cause an automatic end of command as soon as a command letter is entered. (Normally, commands must be terminated by a <RETURN>.) |
| -s | Print all messages and prompts in standout mode (usually inverse video). |

<i>+linenumber</i>	Start up at <i>linenumber</i> .
<i>+/pattern/</i>	Start up at the first line containing the regular expression <i>pattern</i> .

COMMANDS

The responses that you may type when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address.

The perusal commands and their defaults are as follows:

(+1)< <i>newline</i> > or < <i>blank</i> >	Display one page. The address is specified in pages.
(+1) l	With a relative address, simulate scrolling of the screen, forward or backward, the number of lines specified. With an absolute address, print a screenful beginning at the specified line.
(+1) d or ^D	Simulate scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or ^L	Redisplay the current page of text.
\$	Display the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed** (1) are available. They must always be terminated by a <*newline*>, even if the **-n** option is specified.

<i>i/pattern/</i>	Search forward for the <i>i</i> th (default <i>i</i> =1) occurrence of <i>pattern</i> . Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.
-------------------	--

i^*pattern*
i?*pattern*?

Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The caret (^) notation is useful for terminals that cannot handle a question mark (?) properly.

After searching, *pg* normally displays the line found at the top of the screen. This can be modified by appending *m* or *b* to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix *t* can be used to restore the original situation.

You can modify the perusal environment with the following commands:

<i>in</i>	Begin perusing the <i>i</i> th next file in the command line. The <i>i</i> is an unsigned number; default value is 1.
<i>ip</i>	Begin perusing the <i>i</i> th previous file in the command line. The <i>i</i> is an unsigned number; default value is 1.
<i>iw</i>	Display another window of text. If <i>i</i> is present, set the window size to <i>i</i> .
<i>s filename</i>	Save the input in the named file. Only the current file being perused is saved. The white space between the <i>s</i> and <i>filename</i> is optional. This command must always be terminated by a <RETURN>, even if the <i>-n</i> option is specified.
<i>h</i>	Help by displaying an abbreviated summary of available commands.
<i>q</i> or <i>Q</i>	Quit <i>pg</i> .
<i>!command</i>	Pass <i>command</i> to the shell whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a <RETURN>, even if the <i>-n</i> option is specified.

At any time when output is being sent to the terminal, you can quit (normally CTRL/D) or interrupt (normally CTRL/I) the process. If you do so, the *pg* program stops sending output, and displays the prompt. You may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, **pg** acts like **cat(1)**, except that **pg** prints a header before each file (if there is more than one file).

CAUTIONS

If you do not set terminal tabs every eight positions, **pg** may produce undesirable results.

When using **pg** as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

While waiting for terminal input, **pg** responds to **BREAK**, **DEL**, and **^** by terminating execution. Between prompts, however, these signals interrupt the filter's current task and place the user in prompt mode. Use them with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

EXAMPLE

news | pg -p "(Page d):" *Use pg to help read system news.*

FILES

<i>/usr/lib/terminfo/?/*</i>	terminal information database
<i>/tmp/pg*</i>	temporary file when input is from a pipe

RELATED INFORMATION

crypt(1)
ed(1)
grep(1)
terminfo(4)

NAME

pr – print files

USAGE

pr [*options*] [*files*]

DESCRIPTION

Pr prints the named *files* on the standard output. If you specify a dash (–) for *file*, or if you specify no files at all, the standard input is assumed.

By default, **pr** separates each listing into pages, each headed by the page number, a date and time, and the name of the file. Also by default, columns are of equal width, separated by at least one space; lines that do not fit are truncated.

If the standard output is associated with a terminal, **pr** withholds error messages until printing is complete. The options below may appear singly or be combined in any order.

OPTIONS

- | | |
|-------------|--|
| +k | Begin printing with page <i>k</i> (the default is one). |
| –k | Produce <i>k</i> -column output (the default is one). The options –e and –i are assumed for multicolumn output. |
| –a | Print multicolumn output across the page. |
| –m | Merge and print all files simultaneously, one per column (override the –k , and –a options). |
| –d | Doublespace the output. |
| –eck | Expand <i>input</i> tabs to character positions <i>k</i> +1, 2* <i>k</i> +1, 3* <i>k</i> +1, etc. If <i>k</i> is zero or is omitted, the default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If <i>c</i> (any nondigit character) is given, it is treated as the input tab character (the default for <i>c</i> is the tab character). |
| –ick | In <i>output</i> , replace white space wherever possible by inserting tabs to character positions <i>k</i> +1, 2* <i>k</i> +1, 3* <i>k</i> +1, etc. If <i>k</i> is 0 or is omitted, the default tab settings at every eighth position are assumed. If <i>c</i> (any nondigit character) is given, it is treated as the output tab character (the default for <i>c</i> is the tab character). |
| –nck | Provide <i>k</i> -digit line numbering (the default for <i>k</i> is five). The number occupies the first <i>k</i> +1 character positions of each column of normal output or each line of –m output. If <i>c</i> (any nondigit character) is given, it is appended to the line number to separate it from whatever follows (the default for <i>c</i> is a tab). |

- wk** Set the width of a line to *k* character positions (the default is 72 for equal-width multicolumn output; there is no limit otherwise).
- ok** Offset each line by *k* character positions (the default is zero). The number of character positions per line is the sum of the width and offset.
- lk** Set the length of a page to *k* lines (the default is 66).
- h** Use the next argument as the header to be printed instead of the filename.
- p** Pause before beginning each page if the output is directed to a terminal (**pr** rings the bell at the terminal and waits for a carriage return).
- f** Use a form feed character for new pages (the default is to use a sequence of line feeds). Pause before beginning the first page if the standard output is associated with a terminal.
- r** Print no diagnostic reports on failure to open files.
- t** Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- sc** Separate columns by the single character *c* instead of by the appropriate number of spaces (the default for *c* is a tab).

EXAMPLES

To print *file1* and *file2* as a doublespaced, three-column listing headed by "file list", use the following command:

```
pr -3dh "file list" file1 file2
```

Use this to write *file1* on *file2*, expanding tabs to columns 10, 19, 28, 37, ...:

```
pr -e9 -t < file1 > file2
```

RELATED INFORMATION

cat(1).

NAME

`prs` – print an SCCS file

USAGE

`prs [-d[dataspec]] [-r [SID]] [-e] [-l] [-c [date-time]] [-a] files`

DESCRIPTION

`Prs` prints, on the standard output, parts or all of an SCCS file in a user-supplied format. If a directory is named, `prs` behaves as though each file in the directory is specified as a named file, except that it silently ignores non-SCCS and unreadable files. If a dash (-) is given in place of a filename, `prs` reads the standard input, taking each line to be the name of an SCCS file or directory to be processed. Options to `prs` may appear in any order. Each argument applies independently to each named file.

OPTIONS

- | | |
|-----------------------------------|---|
| <code>-d[<i>dataspec</i>]</code> | Specify the output data specification. The <i>dataspec</i> is a string consisting of SCCS file <i>data keywords</i> interspersed with optional user-supplied text. |
| <code>-r[<i>SID</i>]</code> | Specify the CCS Identification (SID) string of the delta for which information is desired. If you do not specify an SID, then <code>prs</code> assumes the SID to be that of the most recently created delta. |
| <code>-e</code> | Request information for all deltas created <i>earlier</i> than and including the delta designated via the <code>-r</code> keyletter or the date given by the <code>-c</code> option. |
| <code>-l</code> | Request information for all deltas created <i>later</i> than and including the delta designated via the <code>-r</code> keyletter or the date given by the <code>-c</code> option. |
| <code>-c[<i>date-time</i>]</code> | Specify <i>date-time</i> as cutoff for requesting information. This cutoff <i>date-time</i> appears in the following form:

<div style="text-align: center;">YY[MM[DD[HH[MM[SS]]]]]</div>
Units omitted from the <i>date-time</i> default to their maximum possible values; that is, <code>-c8502</code> is equivalent to <code>-c850228235959</code> . Any number of non-numeric characters may separate the various two-digit pieces of the <i>cutoff</i> date in the following form:
<code>-c85/2/2 9:22:25</code> . |
| <code>-a</code> | Request printing of information for both removed (delta type = R) and existing (delta type = D) deltas. Refer to <code>rmdel(1)</code> for more information. If you do not specify the <code>-a</code> keyletter, <code>prs</code> only provides information on existing deltas. |

DATA KEYWORDS

Data keywords specify those parts of an SCCS file to be retrieved and output. All parts of an SCCS file have an associated data keyword. Refer to `sccsfile(4)` for more information about the structure of these file types. There is no limit on the number of times a data keyword may appear in a *dataspec*.

Prs prints the user-supplied text, and appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.

A tab is specified by `\t`, and a carriage return/newline is specified by `\n`. The default data keywords are:

```
“:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:”
```

SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	:R::L::B::S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S

SCCS Files Data Keywords (Contd.)

<i>Keyword</i>	<i>Data Item</i>	<i>File Section</i>	<i>Value</i>	<i>Format</i>
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	yes or no	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of <i>what</i> (1) string	N/A	:Z::M:\t:I:	S
:A:	A form of <i>what</i> (1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	<i>what</i> (1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

Using the following command,

```
prs -d"Users and/or user IDs for :F: are:0UN:" s.file
```

may produce these results on the standard output:

Users and/or user IDs for s.file are: xyz 131 abc

Furthermore, if you type this command line:

```
prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
```

the following may appear on the standard output:

Newest delta for pgm main.c: 3.7 Created 85/12/1 By cas

A simple command line without options, such as **prs s.file**, may produce the following on the standard output:

D 1.1 85/12/1 00:00:00 cas 1 000000/00000/00000

MRs:

bl78-12345

bl79-54321

COMMENTS:

this is the comment line for s.file initial delta

for each delta table entry of the "D" type.

FILES

/tmp/pr?????

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

admin(1)

delta(1)

get(1)

help(1)

sccsfile(4)

DOMAIN/IX User's Guide

NAME

ps – report process status

USAGE

ps [*options*]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, it prints information about processes associated with the current user ID (except for *pid 1*, usually the display manager or server process manager). The output consists of a short listing containing only the process ID, user ID, cumulative execution time, and the command name. Otherwise, **ps** controls the displayed information by the selection of *options*.

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

OPTIONS

- e** Print information about all processes.
- d** Print information about all processes, except process group leaders.
- a** Print information about all processes, except process group leaders and processes not associated with a terminal.
- f** Generate a *full* listing. See below for the meaning of columns in a full listing.
- l** Generate a *long* listing. See below.
- p proclst** Restrict the listing to data about processes whose process ID numbers are given in *proclst*.
- u uidlist** Restrict the listing to data about processes whose user ID numbers or log-in names are given in *uidlist*. In the listing, the numerical user ID is printed unless the **-f** option is used, in which case the log-in name is printed.
- g grplist** Restrict listing to data about processes whose process group leaders are given in *grplist*.
- N** Print AEGIS process name.

OUTPUT INTERPRETATION

The column headings and the meaning of the columns in a **ps** listing are given below. The letters *f* and *l* indicate the option (*full* or *long*) that causes the corresponding heading to appear; *all* means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:

- 1 in memory
- 2 system process
- 20 being traced by another process

S (l) The state of the process:

- S sleeping
- W waiting
- R running
- T stopped

UID (f,l) The user ID number of the process owner; the log-in name is printed under the **-f** option.

PID (all) The process ID of the process; you can kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process; higher numbers mean lower priority.

NI (l) Nice value; used in priority computation.

ADDR (l) The memory address of the process, if resident; otherwise, the disk address.

SZ (l) The size, in blocks, of the core image of the process.

STIME (f) Starting time of the process.

TIME (all) The cumulative execution time for the process.

CMD (all) The command name; the full command name and its arguments are

printed under the `-f` option.

CAUTIONS

Data can change drastically while `ps` is running. It is important to realize that the picture it presents at any given point is only a close approximation to reality.

Some data printed for defunct processes are irrelevant.

RELATED INFORMATION

`kill(1)`, `nice(1)`.

NAME

ptx – permuted index

USAGE

ptx [*options*] [*input* [*output*]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). First, it does the permutation, generating one line for each keyword in an input line; then it rotates the keyword to the front and sorts the permuted file; and finally, it rotates the sorted lines so the keyword comes at the middle of each line.

Ptx output appears in the following form:

.xx "tail" "before keyword" "keyword and after" "head"

The .xx shown above is assumed to be an `nroff(1)` or `troff(1)` macro that you provide. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

OPTIONS

- f Fold upper- and lowercase letters for sorting.
- t Prepare the output for the phototypesetter.
- w *n* Use *n* as the length of the output line. The default line length is 72 characters for `nroff(1)` and 100 for `troff(1)`.
- g *n* Use *n* as the number of characters to reserve for each gap among the four parts of the line as finally printed. The default gap is three characters.
- o *only* Use as keywords only the words given in the *only* file.
- i *ignore* Do not use as keywords any words given in the *ignore* file. If the -i and -o options are missing, use `/usr/lib/eign` as the *ignore* file.
- b *break* Use the characters in the *break* file to separate words. Tab, newline, and space characters are *always* used as break characters.
- r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a fifth field on each output line.

CAUTIONS

Line length counts do not account for overstriking or proportional spacing.

Because **ptx** uses tildes internally, lines containing them do not print correctly.

FILES

/bin/sort

/usr/lib/eign

RELATED INFORMATION

nroff(1), **troff(1)**, **mm(5)**.

NAME

pwd – working directory name

USAGE

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

DIAGNOSTICS

“can’t get working directory” indicates possible network or file system trouble.

RELATED INFORMATION

cd(1).

NAME

ratfor – rational FORTRAN dialect

USAGE

ratfor [*options*] [*files*]

DESCRIPTION

Ratfor converts a rational dialect of FORTRAN into ordinary FORTRAN. It provides control flow constructs essentially identical to those in C, as well as simplified syntax to make programs easier to read and write. These constructs are described below:

statement grouping:

{ *statement*; *statement*; *statement* }

decision-making:

if (*condition*) *statement* [else *statement*]

switch (*integer value*) {

case *integer*: *statement*

...

[default:] *statement*

}

loops:

while (*condition*) *statement*

for (*expression*; *condition*; *expression*) *statement*

do *limits* *statement*

repeat *statement* [until (*condition*)]

break

next

free form input:

multiple statements/line; automatic continuation

comments:

this is a comment.

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return expression to caller from function:

return (*expression*)

define:

define *name replacement*

include:

include *file*

OPTIONS

- h** Turn quoted strings into 27H constructs.
- C** Copy comments to the output and attempt to format it neatly.
- 6x** Make the continuation character *x* and place it in column six. Normally, continuation lines are marked with an ampersand (&) in column one.

RELATED INFORMATION

DOMAIN/IX Support Tools Guide

NAME

regcmp – regular expression compile

USAGE

regcmp [-] *files*

DESCRIPTION

Regcmp compiles the regular expressions in *file* and places the output in *file.i* (unless the - option is used, causing the output to be placed in *file.c*).

The format of entries in *file* is a name (C variable) followed by one or more blanks, and then a regular expression enclosed in double quotes.

Regcmp, in most cases, precludes the need for calling **regcmp** (3) from C programs. This saves on both execution time and program size.

The output of **regcmp** is C source code. Compiled regular expressions are represented as *extern char* vectors. Therefore, *file.i* files may be *included* into C programs, or *file.c* files may be compiled and later loaded.

In the C program that uses the **regcmp** output, *regex(abc,line)* applies the regular expression named *abc* to *line*.

EXAMPLES

name "[A-Za-z][A-Za-z0-9_]*"\$0"

telno "\{0,1\}([2-9][01][1-9])\$0\{0,1\} *"
"([2-9][0-9]{2})\$1[-]{0,1}"
"([0-9]{4})\$2"

In the C program that uses the **regcmp** output,

regex(telno, line, area, exch, rest)

applies the regular expression named *telno* to *line*.

DIAGNOSTICS

Self-explanatory.

RELATED INFORMATION

regcmp(3).

NAME

rm, rmdir – remove files or directories

USAGE

rm [**-fri**] *file* ...
rmdir *dir* ...

DESCRIPTION

Rm removes the entries for one or more *files* from a directory. If an entry was the last link to the file, the file is destroyed. Removing a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with y, the file is deleted; otherwise, the file remains. No questions are asked if the standard input is not a terminal.

Rmdir removes entries for the named directories, which must be empty.

OPTIONS

- f** Suppress questions concerning removals.
- r** Recursively delete the entire contents of the specified directory, and the directory itself, without printing any error comments.
- i** Use interactive mode. Ask whether or not to delete each file. If this option is used with the **-r** option, **rm** also asks whether or not to examine each directory.

CAUTIONS

It is forbidden to remove the file “..” merely to avoid the antisocial consequences of inadvertently doing something such as:

rm -r .*

DIAGNOSTICS

Generally self-explanatory.

RELATED INFORMATION

unlink(2).

NAME

rmdel – remove a delta from an SCCS file

USAGE

rmdel -rSID [*file ...*] [*directory*]

DESCRIPTION

Rmdel removes the delta (specified by the SCCS IDentification number, or SID) from each named SCCS *file*. The delta to be removed must be the newest (most recent) one in its branch in the delta chain. The SID specified must *not* be that of a version being edited for the purpose of making a delta. Thus, if a *p-file* exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*. Refer to **get**(1) for information about the creation of *p-files*.

If you give a directory as an argument, **rm**del behaves as though each file in the directory were specified as a named file, except that it silently ignores any non-SCCS and unreadable files. If you supply a dash (-) in place of a filename, the standard input is read. Each line of the standard input is taken to be the name of an SCCS file to be processed; again, non-SCCS files and unreadable files are silently ignored.

Permission to remove a delta is granted to the person who created it, as well as to the owner of the associated file and directory.

FILES

x.file	see delta (1)
z.file	see delta (1)

DIAGNOSTICS

Use **help**(1) for explanations.

RELATED INFORMATION

delta(1), **get**(1), **help**(1), **prs**(1), **scsfile**(4).

NAME

sact – print current SCCS file editing activity

USAGE

sact *file* ...

DESCRIPTION

Sact informs you of any impending deltas to a named SCCS *file*. This information is most helpful if you mistakenly execute a **get(1)** command using the **-e** option without a subsequent execution of **delta(1)**. If a directory is named on the command line, **sact** behaves as though each file in the directory were specified as a named file, except that it ignores non-SCCS and unreadable files.

If you specify a dash (**-**) instead of a filename, **sact** reads the standard input, interpreting each line as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces. These fields specify the following information:

Field 1	SID of an existing delta in the SCCS file to which changes will be made to create the new delta.
Field 2	SID for the new delta to be created.
Field 3	Logname of the user who will make the delta (i.e., the person who executed a get for editing).
Field 4	Date that get -e was executed.
Field 5	Time that get -e was executed.

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

delta(1), **get(1)**, **unget(1)**.

NAME

sccsdiff – compare two versions of an SCCS file

USAGE

sccsdiff *-rSID1 -rSID2 [-p] [-sn] files*

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. You may specify any number of SCCS files, but arguments will apply to all files.

OPTIONS

- rSID?** *SID1* and *SID2* specify the deltas of an SCCS file to be compared. Versions are passed to **bdiff(1)** in the order given.
- p** Pipe output for each file through **pr(1)**.
- sn** Specify *n* as the file segment size that **bdiff** will pass to **diff(1)**. This is useful when **diff(1)** fails due to a high system load.

FILES

/tmp/get????? temporary files

DIAGNOSTICS

Use **help (1)** for explanations.

“ *file* ”: No differences” The two versions are the same.

RELATED INFORMATION

bdiff(1), **get(1)**, **help(1)**, **pr(1)**.

NAME

sdiff – side-by-side difference program

USAGE

sdiff [*options ...*] *file1 file2*

DESCRIPTION

Sdiff uses the output of **diff(1)** to produce a side-by-side listing of two files to indicate those lines which differ. Each line of the two files is printed with a blank gutter between them if the lines are identical, a less-than character (<) in the gutter if the line only exists in *file1*, a greater-than character (>) in the gutter if the line only exists in *file2*, and a pipe character (|) for differing lines.

The following illustrates the use of these characters:

x		y
a		a
b	<	
c	<	
d		d
	>	c

OPTIONS

- w *n*** Use the next argument, *n*, as the width of the output line. The value of *n* must be greater than 20. The default line length is 130 characters.
- l** Only print the left side of any lines that are identical.
- s** Do not print identical lines.
- o *output*** Use *output* as the name of a third file created as a user-controlled merging of *file1* and *file2*. Copy identical lines of *file1* and *file2* to *output*. Print sets of differences, as produced by **diff(1)**, where a set of differences share a common gutter character. After printing each set of differences, prompt the user with a percent sign (%) and wait for one of the following user-typed commands:

- l** Append the left column to the output file.
- r** Append the right column to the output file.
- s** Turn on silent mode; do not print identical lines.
- v** Turn off silent mode.
- e l** Call the editor with the left column.
- e r** Call the editor with the right column.

- e b Call the editor with the concatenation of left and right.
- e Call the editor with a zero length file.
- q Exit from the program.

On exit from the editor, concatenate the resulting file on the end of the *output* file.

RELATED INFORMATION

diff(1), ed(1).

NAME

sed – stream editor

USAGE

sed [*-n*] [*-e script*] [*-f sfile*] [*files*]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, sed cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under *-n*) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

OPTIONS

- e script* Specify *script* as a set of commands with which to edit the named files. If there is just one *-e* option and no *-f* options, the flag *-e* may be omitted.
- f sfile* Cause the script to be taken from file *sfile*. This option accumulates.
- n* Suppress the default output.

ADDRESSES

An *address* is either a decimal number that counts input lines cumulatively across files, a **\$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of ed(1) with the modifications explained in the following paragraphs.

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address *\abc\xdefx*, the second *x* stands for itself, so that the regular expression is *abcxdef*.

The escape sequence *\n* matches a newline *embedded* in the pattern space.

A period (.) matches any character except the *terminal* newline of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Consequently, the process is repeated, looking again for the first address.

You can apply editing commands to nonselected pattern spaces only by using the negation function (!).

FUNCTIONS

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with a backslash (\) to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a\

text Append. Place *text* on the output before reading the next input line.

(2) b *label* Branch to the colon (:) command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) c\

text Change. Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) g Replace the contents of the pattern space by the contents of the hold space.

(2) G Append the contents of the hold space to the pattern space.

(2) h Replace the contents of the hold space by the contents of the pattern

space.

- (2)H Append the contents of the pattern space to the hold space.
- (1)\
text Insert. Place *text* on the standard output.
- (2)l List the pattern space on the standard output in an unambiguous form. Nonprinting characters are spelled in two-digit ASCII and long lines are folded.
- (2)n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
- (2)N Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
- (2)p Print. Copy the pattern space to the standard output.
- (2)P Copy the initial segment of the pattern space through the first newline to the standard output.
- (1)q Quit. Branch to the end of the script. Do not start a new cycle.
- (2)r *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.
- (2)s/ *regular expression* / *replacement* / *flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of a slash (/). For a more complete description, refer to ed(1). *Flags* is zero or more of:
 - n* *n*=1 through 512. Substitute for just the *n*th occurrence of the *regular expression*.
 - g* Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
 - p* Print the pattern space if a replacement was made.
- w wfile*
Write. Append the pattern space to *wfile* if a replacement was made.
- (2)t *label* Test. Branch to the colon (:) command bearing the *label* if any

substitutions have been made since the most recent reading of an input line or execution of a *t*. If *label* is empty, branch to the end of the script.

- (2) *w wfile* Write. Append the pattern space to *wfile*.
- (2) *x* Exchange the contents of the pattern and hold spaces.
- (2) *y/string1/string2/*
Transform. Replace all occurrences of characters in *string1* with the corresponding characters in *string2*. The lengths of *string1* and *string2* must be equal.
- (2) *! function*
Apply the *function* (or group, if *function* is { }) only to lines *not* selected by the address(es).
- (0) *: label* This command does nothing; it bears a *label* to which *b* and *t* commands branch.
- (1) *=* Place the current line number on the standard output as a line.
- (2) *{* Execute the following commands through a matching *}* only when the pattern space is selected.
- (0) An empty command is ignored.
- (0) *#* If a pound sign (*#*) appears as the first character on any line of a script file, then that entire line is treated as a comment, with one exception. On the first line of the script only, if the character after the pound sign is an "n", then the default output will be suppressed. The rest of the line after *#n* is also ignored. A script file must contain at least one noncomment line.

RELATED INFORMATION

awk(1), *ed*(1), *grep*(1).

NAME

sh, rsh – the standard/restricted Bourne Shell (command programming language)

USAGE

sh [*args*]
rsh [*args*]

DESCRIPTION

The Bourne Shell, **/bin/sh**, is a command programming language that executes commands read from a terminal or a file.

Rsh is a restricted version of this shell. It is used to set up log-in names and execution environments whose capabilities are more controlled than those of the standard Shell. **Rsh** operates in much the same manner as **sh**, except that you cannot do the following when using the restricted shell: 1) change directories (i.e., execute a **cd(1)** command), 2) set the value of the **\$PATH** variable, 3) specify path or command names containing a slash (/), or 4) redirect output (> and >>). These restrictions are enforced after your *.profile* is interpreted.

When a command to be executed is found to be a shell procedure, **rsh** invokes **sh** to execute it. **Rsh** assumes that you do not have write and execute permissions in the same directory. The net effect of these rules is that the writer of the *.profile* has complete control over your actions, by performing guaranteed set-up actions and leaving you in an appropriate directory (probably *not* the log-in directory).

The system administrator often sets up a directory of commands (i.e., */usr/rbin*) that **rsh** can safely invoke.

DEFINITIONS

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the following characters: *, @, #, ?, -, \$, and !.

A *simple-command* is a sequence of nonblank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Usually, the remaining words are passed as arguments to the invoked command (exceptions are noted under **COMMANDS**). The command name is passed as argument 0. Refer to **exec(2)** for additional information about this. The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally. See **signal(2)** for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by a pipe character (|) or, for historical compatibility, a caret (^). The standard output of each command but the last is connected by a **pipe(2)** to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by `;`, `&`, `&&`, or `|`, and optionally terminated by `;` or `&`. Of these four symbols, `;` and `&` have equal precedence, which is lower than that of `&&` and `|`. The symbols `&&` and `|` also have equal precedence. A semicolon (`;`) causes sequential execution of the preceding pipeline; an ampersand (`&`) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol `&&` (`|`) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semi-colons, to delimit commands.

COMMANDS

The commands used by `sh` are either simple commands or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

The standard output from a command enclosed in a pair of grave accents (```) may be used as part or all of a word; trailing new-lines are removed.

for *name* **in** *word ...* **do** *list* **done**

Each time a **for** command is executed, set *name* to the next *word* taken from the **in** *word* list. If **in** *word ...* is omitted, then the **for** command executes the *do list* once for each positional parameter set. Execution ends when there are no more words in the list.

case *word* **in** *pattern | pattern ...* *list* **;; ... esac**

Execute the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see FILENAME GENERATION) except that a slash, a leading period, or a period immediately following a slash need not be matched explicitly.

if *list* **then** *list* **elif** *list* **then** *list ...* **else** *list* **fi**

Execute the *list* following **if**. If it returns a zero exit status, then execute the *list* following the first **then**. Otherwise, execute the *list* following **elif** and, if its value is zero, execute the *list* following the next **then**. Failing that, execute the *else list*. If no *else list* or *then list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

Repeatedly execute the *while list* and, if the exit status of the last command in the *list* is zero, execute the *do list*; otherwise, terminate the loop. If no commands in the *do list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{ *list*;

Simply execute *list*. Note the space after the left brace. This is required.

name () { *list*;

Define a function referenced by *name*. The body of the function is the *list* of commands between { and }. Note the space after the left brace. This is required. Execution of functions is described below (see EXECUTION).

The following words are only recognized as the first word of a command and when they are not quoted:

if then else elif fi case esac for while until do done { }

COMMENTS

In most cases, a word beginning with a pound sign (#) causes that word and all the following characters up to a newline to be ignored. This is not true when the pound sign is used in a shell script to specify which shell the script should run in, e.g., *#!/bin/sh* or *#!/com/sh*.

PARAMETERS

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the <i>set</i> command.
?	The decimal value returned by the last synchronously-executed command.
\$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME	The default argument (home directory) for the <i>cd</i> command.
PATH	The search path for commands (see EXECUTION below). You may not change PATH if executing under <i>rsh</i> .

CDPATH	The search path for the <code>cd</code> command.
MAIL	If this parameter is set to the name of a mail file <i>and</i> the <code>MAILPATH</code> parameter is not set, the shell informs you of the arrival of mail in the specified file.
MAILCHECK	This parameter specifies how often (in seconds) the shell checks for the arrival of mail in the files specified by the <code>MAILPATH</code> or <code>MAIL</code> parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell checks before each prompt.
MAILPATH	A list of filenames separated by a colon (:). If this parameter is set, the shell informs you of the arrival of mail in any of the specified files. Each filename can be followed by a percent sign (%) and a message that is printed when the modification time changes. The default message is <i>you have mail</i> .
PS1	Primary prompt string, by default a pound sign (#).
PS2	Secondary prompt string, by default a greater-than sign (>).
IFS	Internal field separators, normally <i>space</i> , <i>tab</i> , and <i>newline</i> .
SHELL	When the shell is invoked, it scans the environment for this name (see <code>ENVIRONMENT</code> below). If it is found and there is an "r" in the filename part of its value, the shell becomes a restricted shell.

The shell gives default values to `PATH`, `PS1`, `PS2`, `MAILCHECK`, and `IFS`. `HOME` and `MAIL` are set by `login(1)`.

PARAMETER SUBSTITUTION

The dollar sign character (\$) introduces substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by set. Keyword parameters (also known as variables) may be assigned values by writing:

name=value

name=value

...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

`${parameter}`

Substitute the value, if any, of the parameter. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is an asterisk (*) or an at sign (@), all the positional parameters, starting with \$1, are substituted (separated by spaces). Parameter \$0 is set from argument zero when the shell is invoked.

`${parameter:-word}`

If *parameter* is set and is non-null, substitute its value; otherwise, substitute *word*.

`${parameter:=word}`

If *parameter* is not set or is null, set it to *word*; substitute the value of the parameter. Positional parameters may not be assigned to in this way.

`${parameter:?word}`

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, print the message "parameter null or not set".

`${parameter:+word}`

If *parameter* is set and is non-null, substitute *word*; otherwise, substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-'pwd'}
```

If the colon (:) is omitted from the above expressions, the shell only checks to see whether *parameter* is set or not.

BLANK INTERPRETATION

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in `IFS`) and split into distinct arguments where such characters are found. Explicit null arguments (enclosed in double or single quotes) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

FILENAME GENERATION

Following substitution, each command *word* is scanned for asterisks, question marks, and periods. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. A period at the start of a filename or immediately following a slash (/), as well as the slash character itself, must be matched explicitly. The characters and their meanings are:

- * Matches any string, including the null string.
 - ? Matches any single character.
 - [...]
- Matches any one of the enclosed characters. A pair of characters separated by a dash (-) matches any character lexically between the pair, inclusive. If the first character following the opening bracket is an exclamation point (!), any character not enclosed is matched.

QUOTING

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & () | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a backslash (\). The pair `\newline` is ignored. All characters enclosed between a pair of single quote marks, except a single quote, are quoted. Inside double quote marks, parameter and command substitution occurs and a backslash quotes the following characters: \, \, ", and \$. A "\$*" is equivalent to "\$1 \$2 ...", while "\$@" is equivalent to "\$1" "\$2"

PROMPTING

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of PS2) is issued.

INPUT/OUTPUT

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command, or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

< <i>word</i>	Use file <i>word</i> as standard input (file descriptor 0).
> <i>word</i>	Use file <i>word</i> as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.
>> <i>word</i>	Use file <i>word</i> as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<- <i>word</i>	Read the shell input up to a line that is the same as <i>word</i> , or to an end-of-file. The resulting document becomes the standard input. If any character of <i>word</i> is quoted, no interpretation is placed upon the characters of the document. Otherwise, parameter and command substitution occurs; (unescaped) <code>\newline</code> is ignored; and <code>\</code> must be used to quote a backslash, a dollar sign, a grave accent, and the first character of <i>word</i> . If a dash (-) is appended to <<, all leading tabs are stripped from <i>word</i> and from the document.
<& <i>digit</i>	Use the file associated with file descriptor <i>digit</i> as standard input. Do

the same for the standard output, using `>&digit`.
`<&-` Close the standard input. Do the same for the standard output, using `>&-`.

If any of the above is preceded by a digit, the file descriptor that will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file `xxx`. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., `xxx`). If the order of redirections is reversed, file descriptor 2 is associated with the terminal (assuming file descriptor 1 had been associated with it) and file descriptor 1 is associated with file `xxx`.

If a command is followed by `&` the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for executing a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

ENVIRONMENT

The *environment* is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. Refer to `environ(5)` for more general information. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If you modify the value of any of these parameters or create new parameters, none of these affects the environment unless the `export` command is used to bind the shell's parameter to the environment (refer to `set -a`). You may remove a parameter from the environment with the `unset` command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by `unset`, plus any modifications or additions, all of which must be noted in `export` commands.

You can augment the environment for any *simple command* by prefixing it with one or more assignments to parameters. Thus,

```
TERM=450 cmd
```

and

```
(export TERM; TERM=450; cmd)
```


are equivalent (as far as the execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

SIGNALS

The **INTERRUPT** and **QUIT** signals for an invoked command are ignored if the command is followed by an ampersand (&). Otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11. (Refer to the **trap** command below.)

EXECUTION

Each time a command is executed, the shell carries out the above substitutions. If the command name matches one of the special commands listed below, it is executed in the shell process. If the command name does not match a special command, but matches the name of a defined function, the function is executed in the shell process. Note how this differs from the execution of shell procedures. The positional parameters \$1, \$2, ... are set to the arguments of the function. If the command name does not match a special command or the name of a defined function, a new process is created and an attempt is made to execute the command via **exec(2)**.

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a backslash, the search path is not used. Such commands are not executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell, to help avoid unnecessary invocations of **exec** later. If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

SPECIAL COMMANDS

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

- :** Do nothing but return an exit code of zero.
- . *file*** Read and execute commands from *file* and return. Use the search path specified by **PATH** to find the directory containing *file*.
- break [*n*]**
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.
- continue [*n*]**
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*th enclosing loop.
- cd [*arg*]**
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). The current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a slash (/), the search path is not used. Otherwise, each directory in the path is searched for *arg*. The **cd** command may not be executed by **rsh**.

If **cd** locates a directory via **CDPATH**, it typically reports the new pathname. If, however, the path happens to include a softlink, this can be somewhat misleading. For example, if **CDPATH** = **robert**, and a user with the log-in name *joe* does a **cd joe** from his current directory, the following occurs: 1) **cd** looks for a *joe* directory in the present working directory. It will not find it, so it 2) looks for a *joe* directory in the **//robert** directory. If one exists, and is a softlink (i.e., **//robert/joe** points off to **//hol/joe**), the **cd** command states that it has changed directories to **//robert/joe** instead of **//hol/joe**.
- echo [*arg* ...]**
Echo arguments. Refer to **echo(1)** for usage and description.
- eval [*arg* ...]**
Read the arguments as input to the shell and execute the resulting command(s).
- exec [*arg* ...]**
Execute the command specified by the arguments in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- exit [*n*]**
Exit the shell with the exit status specified by *n*. If *n* is omitted, the exit status

is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]

Mark the given *names* for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, print a list of all *names* exported in this shell. Function names may *not* be exported.

hash [*-r*] [*name* ...]

For each *name*, determine and remember the location in the search path of the command specified by *name*. If *-r* is also used, the shell forgets all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this is done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* is incremented when the recalculation is done.

newgrp [*arg* ...]

Perform the same function as **exec newgrp** *arg*

pwd Print the current working directory. See **pwd(1)** for usage and description.

read [*name* ...]

Read one line from the standard input and assign the first word to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]

Mark the given *names* as *readonly* and disallow any changes to the values of these *names* by subsequent assignment. If no arguments are given, print a list of all *readonly* names.

return [*n*]

Exit the function with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [*-aefhkntuvx* [*arg* ...]]

- a* Mark variables modified or created for export.
- e* Exit immediately if a command exits with a non-zero exit status.
- f* Disable the filename generation.
- h* Locate and remember function commands as functions are defined (these are normally located when the function is executed).

- k Place all keyword arguments in the environment for a command, not just those that precede the command name.
- n Read commands but do not execute them.
- t Exit after reading and executing one command.
- u Treat unset variables as an error when substituting.
- v Print shell input lines as they are read.
- x Print commands and their arguments as they are executed.
- Do not change any of the flags. This is useful in setting \$1 to -.

Using + rather than - causes these flags to be turned off. These flags may also be used when invoking the shell. The current set of flags may be found in \$-. The remaining arguments are positional parameters assigned, in order, to \$1, \$2, If no arguments are given the values of all names are printed.

shift [*n*]

Rename the positional parameters from \$*n*+1 ... to \$1, If *n* is not given, it is assumed to be 1.

test

Evaluate conditional expressions. See test(1) for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n* ...]

Read the command *arg* and execute it when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent, reset all trap(s) *n* to their original values. If *arg* is the null string, the shell and the commands it invokes will ignore the signal. If *n* is 0, execute the command *arg* on exit from the shell. With no arguments, print a list of commands associated with each signal number.

type [*name* ...]

For each *name*, indicate how it will be interpreted if used as a command name.

ulimit [-fp [*n*]]

With the -f option, impose a size limit of *n* blocks on files written by child processes (files of any size may be read). With no *n* argument, print the current limit. With the -p option, change the pipe size to *n* (UNIX/RT only). If no option is given, -f is assumed.

umask [*nnn*]

Set the user file-creation mask to *nnn*. Refer to umask(2) for more information. If *nnn* is omitted, print the current value of the mask.

unset [*name* ...]

For each *name*, remove the corresponding variable or function. The variables PATH, PS1, PS2, MAILCHECK, and IFS cannot be unset.

wait [*n*]

Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

INVOCATION

If the shell is invoked through `exec(2)` and the first character of argument zero is a dash (-), commands are initially read from `/etc/profile` and from `$HOME/.profile`, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as `/bin/start_sh`. The flags below are interpreted by the shell on invocation only. Unless the `-c` or `-s` flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file.

- `-c string` Read commands from *string*.
- `-s` Read commands from the standard input if this flag is present or no arguments remain. Any remaining arguments specify the positional parameters. Write shell output (except for special commands above) to file descriptor 2.
- `-i` Make the shell interactive if this flag is present or if the shell input and output are attached to a terminal. Ignore `TERMINATE` so that `kill 0` does not kill an interactive shell, and catch and ignore `INTERRUPT` so that `wait` is interruptible. In all cases, the shell ignores `QUIT`.
- `-r` Make the shell restricted if this flag is present.

The remaining flags and arguments are described under the `set` command listed under **SPECIAL COMMANDS**.

CAUTIONS

If you execute a command, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell continues to perform an `exec(2)` on the original command. Use the `hash` command to correct this situation.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is not being used interactively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the `exit` command under **SPECIAL COMMANDS** above).

FILES

<code>/etc/profile</code>	global profile
<code>\$HOME/.profile</code>	user's profile
<code>/tmp/sh*</code>	temp file

/dev/null

bit sink

RELATED INFORMATION

cd(1)
echo(1)
env(1)
login(1)
pwd(1)
test(1)
umask(1)
dup(2)
exec(2)
fork(2)
pipe(2)
signal(2)
umask(2)
wait(2)
profile(4)
environ(5)

DOMAIN/IX User's Guide

NAME

size – print sizes of object files

USAGE

size [*object ...*]

DESCRIPTION

Size prints the number of bytes (in decimal) required by the text, data, and bss portions of each named *object*. It also prints the sum of these figures in decimal. If you fail to specify an object file, *a.out* is used.

NOTE: Object modules in DOMAIN/IX system processing are formatted differently from those normally found in standard UNIX system operations. Thus, the sizes of text, data, and bss fields will be different.

NAME

sleep – suspend execution for an interval

USAGE

sleep *time*

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

(**sleep 105; command**)&

or to execute a command every so often, as in:

```
while true
do
    command
sleep 37
done
```

RELATED INFORMATION

alarm(2), **sleep(3C)**.

NAME

sort – sort and/or merge files

USAGE

sort [-cmu] [-ooutput] [-ykmem] [-zrecsz] [-dfiMnr] [-btx]
[+pos1 [-pos2]] [files]

DESCRIPTION

Sort puts lines of all the named files together, sorts them, and writes the result on the standard output. It reads the standard input if a dash (-) is used instead of a filename or no input files are named.

Sort bases comparisons on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line. In addition, ordering is normally lexicographic by bytes in machine collating sequence.

OPTIONS

- c Ensure that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.
- m Merge only; input files are already sorted.
- u Suppress all but one in each set of lines having equal keys.
- ooutput Use *output* file instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between -o and *output*.
- ykmem Use the value of *kmem* as the amount of kilobytes of memory to be used by the sort, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, -y0 is guaranteed to start with minimum memory. By convention, -y with no argument starts with maximum memory. The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, sort begins using a system default memory size, and continues to use more space as needed.
- zrecsz Record the size of the longest line read in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the -c or -m options, a popular system default size will be used. Lines longer than the buffer size cause abnormal termination. Supplying the actual number of bytes in the longest line to be merged (or some larger value) prevents abnormal termination.
- d Sort in “dictionary” order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

- f** Fold lowercase letters into uppercase.
- i** Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.
- M** Compare as months. Fold the first three nonblank characters of the field to uppercase and compare so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see below).
- n** Sort an initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.
- r** Reverse the sense of comparisons.
- tx** Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).
- b** Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

+pos1 -pos2 Restrict a sort key to one beginning at *pos1* and ending at *pos2*. Include the characters at positions *pos1* and *pos2* in the sort key, provided that *pos2* does not precede *pos1*. A missing *-pos2* means the end of the line. Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line.

By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field. For example, all blanks at the beginning of a line are considered to be part of the first field.

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the **bdfnr** flags. A starting position specified by *+m.n* is

interpreted to mean the $n+1$ st character in the $m+1$ st field. A missing $.n$ means $.0$, indicating the first character of the $m+1$ st field. If the **b** flag is in effect, n is counted from the first nonblank character in the $m+1$ st field; $+m.0b$ refers to the first nonblank character in the $m+1$ st field.

A last position specified by $-m.n$ is interpreted to mean the n th character (including separators) after the last character of the m th field. A missing $.n$ means $.0$, indicating the last character of the m th field. If the **b** flag is in effect n is counted from the last leading blank in the $m+1$ st field; $-m.1b$ refers to the first nonblank character in the $m+1$ st field. When there are multiple sort keys, later keys are compared only after all earlier keys compared equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

To sort the contents of *infile* with the second field as the sort key, issue the following command:

```
sort +1 -2 infile
```

To sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key, use this:

```
sort -r -o outfile +1.0 -1.2
```

To sort, in reverse order, the contents of *infile1* and *infile2*, using the first nonblank character of the second field as the sort key, type the following:

```
sort -r +1.0b -1.1b infile1 infile2
```

To print the password file sorted by the numeric user ID (the third colon-separated field), use this:

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

FILES

/usr/tmp/stm???

DIAGNOSTICS

Sort makes comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the **-c** option.

When the last line of an input file is missing a newline character, sort appends one, prints a warning message, and continues.

RELATED INFORMATION

comm(1), join(1), uniq(1).

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

USAGE

spell [**-v**] [**-b**] [**-x**] [**-l**] [**-i**] [*+local_file*] [*files*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words not present on the spelling list are printed on the standard output. Furthermore, it prints those that are not derivable from words on the spelling list (by applying certain inflections, prefixes, and/or suffixes) on the standard output. If no *files* are named, words are collected from the standard input.

The spelling list is based on many sources. While more random than an ordinary dictionary, it is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies used in biology, medicine, and chemistry is light.

You may specify pertinent auxiliary files by name arguments, indicated below with their default settings (see FILES). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., *thier=thy-y+ier*) that would otherwise pass.

Spell ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructions.

OPTIONS

- v** Print all words not literally in the spelling list. Indicate plausible derivations from the words in the spelling list.
- b** Check British spelling. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon using *-ise* in words such as *standardise*.
- x** Print every plausible stem with = for each word. By default, **spell** follows chains of included files, e.g., *.so* and *.nx* requests made by **troff**(1), unless the names of */usr/lib*.
- l** Follow the chains of *all* included files.
- i** Ignore all chains of included files.
- +local_file** Remove words found in *local_file* from output. *Local_file* is a file that you provide. It comprises a sorted list of words, one per line, used as a

personalized spelling list to supplement the one provided by `spell`.

HASH LIST ROUTINES

Three routines help maintain and check the hash lists used by `spell`. These routines and their specific purposes are:

- hashmake** Read a list of words from the standard input and write the corresponding nine-digit hash code on the standard output.
- spellin *n*** Read *n* hash codes from the standard input and write a compressed spelling list on the standard output. Print information about the hash coding on the standard error.
- hashcheck** Read a compressed *spelling_list* and re-create the nine-digit hash codes for all the words in it. Write these codes on the standard output.

EXAMPLE

The following example creates the hashed spell list *hlist* and checks the result by comparing the two temporary files; they should be equal.

```
cat goodwds | /usr/lib/spell/hashmake | sort -u >tmp1
cat tmp1 | /usr/lib/spell/spellin 'cat tmp1 | wc -l' >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

CAUTIONS

The spelling list's coverage is uneven. You may want to monitor the output for several months to gather local additions. Typically, these are kept in a separate local file that is added to the hashed *spelling_list* via `spellin`.

FILES

<i>D_SPELL</i> = <i>/usr/lib/spell/hlist[ab]</i>	hashed spelling lists, American and British
<i>S_SPELL</i> = <i>/usr/lib/spell/hstop</i>	hashed stop list
<i>H_SPELL</i> = <i>/usr/lib/spell/spellhist</i>	history file
<i>/usr/lib/spell/spellprog</i>	program

RELATED INFORMATION

`deroff(1)`, `eqn(1)`, `sed(1)`, `sort(1)`, `tbl(1)`, `tee(1)`, `troff(1)`.

NAME

spline – interpolate smooth curve

USAGE

spline [options]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by graph(1G).

OPTIONS

The following options are recognized, each as a separate argument:

- a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.
- k The constant k used in the boundary value computation:
$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$
is set by the next argument (default $k = 0$).
- n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).
- p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.
- x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

DIAGNOSTICS

When data is not strictly monotone in x , spline reproduces the input without interpolating extra points.

NOTES

A limit of 1,000 input points is enforced silently.

RELATED INFORMATION

graph(1G).

NAME

split – split a file into pieces

USAGE

split [*-n*] [*file* [*name*]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (the default is 1000 lines) onto a set of output files. The name of the first output file is *name* with *aa* appended. The names of subsequent files also have two characters appended, up to and including *zz*. The total number of output files cannot exceed 676.

Name cannot be longer than 12 characters. If you specify no output name, *x* is the default filename.

If you do not indicate an input file, or if you specify a dash (–) instead, **split** uses the standard input.

RELATED INFORMATION

bfs(1), **csplit(1)**.

NAME

start_csh – start a login C Shell

USAGE

start_csh

DESCRIPTION

Start_csh starts a log-in C Shell (one that reads your *.login* file) Users typically include it in their *~user_data/startup_dm* file to create a login C Shell automatically when they log in. See *cs(1)* for more information about the C shell.

RELATED INFORMATION

cs(1)

NAME

start_sh – run a log-in Bourne Shell

USAGE

cp /bin/start_sh
cp /bin/start_rsh

DESCRIPTION

Start_sh starts a log-in Bourne Shell (one that reads */etc/profile* and your own *.profile*). If invoked as **start_rsh**, it starts a restricted log-in Bourne Shell. Include it in your DM startup script (*user_data/startup_dm.**) to create a Bourne Shell every time you log in. You may also issue this command from the Display Manager, once log-in is complete, to create as many active Bourne Shells as you like. See **sh(1)** for more information about the Bourne Shell.

Note: There is a distinct difference between typing **/bin/start_sh** and **/bin/sh** to the Display Manager. Typing **/bin/start_sh**, lets you start a “log-in”-style Shell even after initial log-in. Thus, **/bin/sh** is invoked and commands are initially read from */etc/profile* and *\$HOME/.profile* (if such files exist).

RELATED INFORMATION

sh(1)

NAME

stat – statistical network useful with graphical commands

USAGE

node-name [*options*] [*files*]

DESCRIPTION

Stat is a collection of command level functions (nodes) that can be interconnected using **sh(1)** to form a statistical network. The nodes reside in */usr/bin/graf* (see **graphics(1G)**). Data is passed through the network as sequences of numbers (vectors), where a number is of the form:

[sign](digits)(.digits)[e[sign]digits]

evaluated in the usual way. Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

Stat node-names are divided into four classes.

<i>Transformers</i> ,	which map input vector elements into output vector elements;
<i>Summarizers</i> ,	which calculate statistics of a vector;
<i>Translators</i> ,	which convert among formats; and
<i>Generators</i> ,	which are sources of definable vectors.

Below is a list of synopses for **stat** nodes. Most nodes accept options indicated by a leading minus (-). In general, an option is specified by a character followed by a value, such as **c5**. This is interpreted as **c := 5** (**c** is assigned 5). The following keys are used to designate the expected type of the value:

c	characters,
i	integer,
f	floating point or integer,
file	file name, and
string	string of characters, surrounded by quotes to include a shell argument delimiter.

Options without keys are flags. All nodes except *generators* accept files as input, hence it is not indicated in the synopses.

Transformers:

abs	[<i>-ci</i>] – absolute value columns (similarly for <i>-c</i> options that follow)
af	[<i>-ci t v</i>] – arithmetic function titled output, verbose
ceil	[<i>-ci</i>] – round up to next integer
cusum	[<i>-ci</i>] – cumulative sum
exp	[<i>-ci</i>] – exponential
floor	[<i>-ci</i>] – round down to next integer
gamma	[<i>-ci</i>] – gamma
list	[<i>-ci dstring</i>] – list vector elements delimiter(s)
log	[<i>-ci bf</i>] – logarithm base
mod	[<i>-ci mf</i>] – modulus modulus
pair	[<i>-ci Ffile xi</i>] – pair elements File containing base vector, x group size
power	[<i>-ci pf</i>] – raise to a power power
root	[<i>-ci rf</i>] – take a root root
round	[<i>-ci pi si</i>] – round to nearest integer, .5 rounds to 1 places after decimal point, significant digits
siline	[<i>-ci if nisf</i>] – generate a line given slope and intercept intercept, number of positive integers, slope
sin	[<i>-ci</i>] – sine
subset	[<i>-af bf ci Ffile ii lf nl np pf si ti</i>] – generate a subset above, below, File with master vector, interval, leave, master con- tains element numbers to leave, master contains element numbers to pick, pick, start, terminate

Summarizers:

bucket	[-ai ci Ffile hf ii lf ni] – break into buckets average size, File containing bucket boundaries, high, interval, low, number Input data should be sorted
cor	[-Ffile] – correlation coefficient File containing base vector
hilo	[- h l o ox oy] – find high and low values high only, low only, option form, option form with x prepended, option form with y prepended
lreg	[-Ffile i o s] – linear regression File containing base vector, intercept only, option form for <i>siline</i> , slope only
mean	[-ff ni pf] – (trimmed) arithmetic mean fraction, number, percent
point	[-ff ni pf s] – point from empirical cumulative density function fraction, number, percent, sorted input
prod	– internal product
qsort	[-ci] – quick sort
rank	– vector rank
total	– sum total
var	– variance

Translators:

bar	[-a b f g ri wi xf xa yf ya ylf yhf] – build a bar chart suppress axes, bold, suppress frame, suppress grid, region, width in percent, x origin, suppress x-axis label, y origin, suppress y-axis label, y-axis lower bound, y-axis high bound Data is rounded off to integers.
hist	[-a b f g ri xf xa yf ya ylf yhf] – build a histogram suppress axes, bold, suppress frame, suppress grid, region, x origin, suppress x-axis label, y origin, suppress y-axis label, y-axis lower bound, y-axis high bound
label	[-b c Ffile h p ri x xu y yr] – label the axis of a GPS file bar chart input, retain case, label File, histogram input, plot input, rotation, x-axis, upper x-axis, y-axis, right y-axis

- pie** **[-b o p pni ppi ri v xi yi]** – build a pie chart
 bold, values outside pie, value as percentage(=100), value as percentage(=i), draw percent of pie, region, no values, x origin, y origin
 Unlike other nodes, input is lines of the form
 [< i e f cc >] value [label]
 ignore (do not draw) slice, explode slice, fill slice, color slice
 c=(black, red, green, blue)
- plot** **[-a b cstring d f Ffile g m ri xf xa xif xhf xlf xni xt yf ya yif yhf ylf yni yt]** – plot a graph
 suppress axes, bold, plotting characters, disconnected, suppress frame, File containing x vector, suppress grid, mark points, region, x origin, suppress x-axis label, x interval, x high bound, x low bound, number of ticks on x-axis, suppress x-axis title, y origin, suppress y-axis label, y interval, y high bound, y low bound, number of ticks on y-axis, suppress y-axis title
- title** **[-b c lstring vstring ustring]** – title a vector or a GPS
 title bold, retain case, lower title, upper title, vector title
- Generators:*
- gas** **[-ci if ni sf tf]** – generate additive sequence
 interval, number, start, terminate
- prime** **[-ci hi li ni]** – generate prime numbers
 high, low, number
- rand** **[-ci hf lf mf ni si]** – generate random sequence
 high, low, multiplier, number, seed

RESTRICTIONS

Some nodes have a limit on the size of the input vector.

RELATED INFORMATION

graphics(1G).

gps(4) in the *UNIX System Programmer Reference Manual*.

NAME

strip – strip symbol and line number information from an object file

USAGE

strip *file(s)*

DESCRIPTION

Strip removes from *file(s)* the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. It is used, for example, to make a fully debugged object module smaller.

The effect of **strip** is the same as use of the **-s** option of **ld(1)**. Programs that have been stripped are difficult or impossible to debug.

FILES

*/tmp/stm.** temporary file

RELATED INFORMATION

ld(1).

NAME

stty – set the options for a terminal

USAGE

stty [**-a**] [**-g**] [*options*]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input. Without arguments, it reports the settings of certain options. With the **-a** argument, it reports all of the option settings; with the **-g** argument, it reports current settings in a form that can be used as an argument to another **stty** command.

Detailed information about the modes listed in the first five groups below may be found in **termio(7)**.

The options listed under COMBINATION MODE OPTIONS are implemented using options in the previous groups. Note that many combinations of options make no sense, but **stty** does no sanity checking.

CONTROL MODE OPTIONS

parenb (-parenb)	Enable (disable) parity generation and detection.
parodd (-parodd)	Select odd (even) parity.
cs5 cs6 cs7 cs8	Select character size.
0	Hang up phone line immediately.
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb	Set terminal baud rate to the number given, if possible. All speeds are not supported by all hardware interfaces.
hupcl (-hupcl)	Hang up (do not hang up) phone connection on the last close.
hup (-hup)	Same as hupcl (-hupcl).
cstopb (-cstopb)	Use two (one) stop bits per character.
cread (-cread)	Enable (disable) the receiver.
clocal (-clocal)	Assume a line without (with) modem control.
loblk (-loblk)	Not supported in DOMAIN/IX.

INPUT MODE OPTIONS

ignbrk (-ignbrk)	Ignore (do not ignore) break on input.
brkint (-brkint)	Signal (do not signal) INTR on break.
ignpar (-ignpar)	Ignore (do not ignore) parity errors.

parmrk (-parmrk)	Mark (do not mark) parity errors.
inpck (-inpck)	Enable (disable) input parity checking.
istrip (-istrip)	Strip (do not strip) input characters to seven bits.
inlcr (-inlcr)	Map (do not map) NL to CR on input.
igncr (-igncr)	Ignore (do not ignore) CR on input.
icrnl (-icrnl)	Map (do not map) CR to NL on input.
iuclic (-iuclic)	Map (do not map) uppercase alphabets to lowercase on input.
ixon (-ixon)	Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1. This is the default in DOMAIN/IX.
ixany (-ixany)	Not supported in DOMAIN/IX.
ixoff (-ixoff)	Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

OUTPUT MODE OPTIONS

opost (-opost)	Postprocess output (do not postprocess output; ignore all other output modes).
olcuc (-olcuc)	Map (do not map) lowercase alphabets to uppercase on output.
onlcr (-onlcr)	Map (do not map) NL to CR-NL on output.
ocrnl (-ocrnl)	Map (do not map) CR to NL on output.
onocr (-onocr)	Do not (do) output CRs at column zero.
onlret (-onlret)	Force the terminal NL to perform (not perform) the CR function.
ofill (-ofill)	Use fill characters (use timing) for delays.
ofdel (-ofdel)	Fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	Not supported in DOMAIN/IX.
nl0 nl1	Select the style of delay for line feeds.
tab0 tab1 tab2 tab3	Not supported in DOMAIN/IX.
bs0 bs1	Not supported in DOMAIN/IX.

ff0 ff1 Not supported in DOMAIN/IX.

vt0 vt1 Not supported in DOMAIN/IX.

LOCAL MODE OPTIONS

isig (-isig) Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWITCH.

icanon (-icanon) Enable (disable) canonical input (ERASE and KILL processing).

xcase (-xcase) Use canonical (unprocessed) upper/lowercase presentation.

echo (-echo) Not supported in DOMAIN/IX. Echo back (do not echo back) every character typed.

echoe (-echoe) Echo (do not echo) the ERASE character as a backspace-space-backspace string. Note: this mode erases the ERASEd character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (-echok) Echo (do not echo) NL after the KILL character.

lfkc (-lfkc) Same as **echok (-echok)**; this option is obsolete.

echonl (-echonl) Echo (do not echo) NL.

noflsh (-noflsh) Disable (enable) flush after INTR, QUIT, or SWITCH.

stwrap (-stwrap) Disable (enable) truncation of lines longer than 79 characters on a synchronous line.

stflush (-stflush) Enable (disable) flush on a synchronous line after every write(2).

stappl (-stappl) Use the application mode (use line mode) on a synchronous line.

CONTROL ASSIGNMENT OPTIONS

control character *c* Set *control character* to *c*, where *control character* is erase, kill, intr, quit, swch, eof, ctab, min, or time. Ctab is used with **-stappl**. Min and time are used with **-icanon**. If *c* is preceded by a caret (^) to escape the Shell, then the value of the corresponding CTRL character is used. A “^?” is interpreted as DEL and “^_” is interpreted as undefined.

line *i* Not supported in DOMAIN/IX.

COMBINATION MODE OPTIONS

evenp or parity	Enable parenb and cs7.
oddp	Enable parenb, cs7, and parodd.
-parity, -evenp, or -oddp	Disable parenb, and set cs8.
raw (-raw or cooked)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output postprocessing).
nl (-nl)	Unset (set) icrnl, onlcr. In addition -nl unsets inlcr, igncr, ocrnl, and onlret.
lcase (-lcase)	Set (unset) xcase, iuclic, and olcuc.
LCASE (-LCASE)	Same as lcase (-lcase).
tabs (-tabs or tab3)	Preserve (expand to spaces) tabs when printing.
ek	Reset the ERASE and KILL characters back to normal # and @.
sane	Reset all modes to some reasonable values.
term	Set all modes suitable for the terminal type <i>term</i> , where <i>term</i> is one of tty33, tty37, vt05, tn300, ti700, or tek.

CAUTIONS

In a Display Manager pad, only the **echo** (-echo) and **raw** (-raw or cooked) options are immediately effective. All other settings are remembered for use when a window is running under control of a VT100 emulator.

RELATED INFORMATION

tabs(1), **ioctl(2)**, **termio(7)**.

NAME

su – become super-user or another user

USAGE

su [-] [*name* [*arg* ...]]

DESCRIPTION

Su allows you to be recognized as another user without logging off. The default user *name* is *root* (i.e., super-user).

Su requires that you supply the appropriate password (unless you are already *root*). If the password is correct, **su** executes a new Shell with the real and effective user ID set to that of the specified user. The new Shell will be the optional program named in the Shell field of the specified user's password file entry, or */bin/sh* if none is specified.

See **passwd(4)** and **sh(1)**

for more information. To restore normal user ID privileges, type an EOF (↑Z) to the new Shell.

Any additional arguments given on the command line are passed to the program invoked as the Shell. When using programs like **sh(1)**, an *arg* of the form **-c string** executes *string* via the Shell, and an *arg* of **-r** gives the user a restricted Shell.

The following statements are true only if the optional program named in the Shell field of the specified user's password file entry is like **sh(1)**. If the first argument to **su** is a dash (-), the environment is changed to what would be expected if you actually logged in as the specified user. This is done by invoking the program used as the Shell with an *arg0* value whose first character is -, thus causing first the system's profile (*/etc/profile*) and then the specified user's profile (*.profile* in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of *\$PATH*, which is set to */bin:/etc:/usr/bin* for *root*.

Note that if the optional program used as the Shell is */bin/sh*, your *.profile* can check *arg0* for **-sh** or **-su** to determine if it was invoked by **login(1)** or **su** respectively. If your program is other than */bin/sh*, then *.profile* is invoked with an *arg0* of *-program* by both **login(1)** and **su**.

All attempts to become another user using **su** are logged in the log file */usr/adm/sulog*.

EXAMPLES

To become user *bin* while retaining your previously exported environment, execute the following:

```
su bin
```

To become user *bin* but change the environment to what would be expected if *bin* had originally logged in, execute:

su - bin

To execute *command* with the temporary environment and permissions of user *bin*, type:

su - bin -c *command args*

FILES

<i>/etc/passwd</i>	system's password file
<i>/etc/profile</i>	system's profile
<i>\$HOME.profile</i>	user's profile
<i>/usr/adm/sulog</i>	log file

RELATED INFORMATION

env(1), login(1), sh(1), passwd(4), environ(5).

NAME

sum – print checksum and block count of a file

USAGE

sum [**-r**] *file*

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named *file*. It also prints the number of blocks in the file. **Sum** is typically used to look for bad spots, or to validate a file communicated over some transmission line.

The **-r** option causes an alternate algorithm to be used in computing the checksum.

DIAGNOSTICS

“Read error” is indistinguishable from end-of-file on most devices. Check the block count.

RELATED INFORMATION

wc(1).

NAME

sync – forces write to disk

USAGE

sync

DESCRIPTION

The sync command executes the sync system primitive. It flushes all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point are saved.

The sync operation is not actually necessary on DOMAIN hardware, because the system buffers are automatically written to disk at shutdown. Nevertheless, we provide it in the interest of ensuring compatibility.

RELATED INFORMATION

sync(2)

NAME

tabs – set tabs on a terminal

USAGE

tabs [*tabspec*] [*+mn*] [*-Ttype*]

DESCRIPTION

Tabs clears any previous tab settings on your terminal, and sets new tab stops according to *tabspec*. The terminal must have hardware tabs that can be remotely set. Tab and margin setting is performed via the standard output.

Four types of tab specifications are accepted for *tabspec*: “canned,” repetitive, arbitrary, and file.

If no *tabspec* is given, the default value is -8 , i.e., UNIX system “standard” tabs. The lowest column number is 1. Note that for tabs, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0 (e.g., the DASI 300, DASI 300s, and DASI 450).

TABSPECS

-code

Give the name of one of a set of “canned” tabs. The legal codes and their meanings are as follows:

- a** 1,10,16,36,72
Assembler, IBM S/370, first format
- a2** 1,10,16,40,72
Assembler, IBM S/370, second format
- c** 1,8,12,16,20,55
COBOL, normal format
- c2** 1,6,10,14,49
COBOL compact format (columns 1 through 6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t-c2 m6 s66 d:>
- c3** 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1 through 6 omitted), with more tabs than -c2. This is the recommended format for COBOL. The appropriate format specification is:

<:t-c3 m6 s66 d:>

-d	1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61,65,69,73,77 DOMAIN - Apollo Display Manager format
-f	1,7,11,15,19,23 FORTRAN
-p	1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61 PL/I
-s	1,10,55 SNOBOL
-u	1,12,20,44 UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

- n A repetitive specification requests tabs at columns $1+n$, $1+2*n$, etc. This type of setting leaves a left margin of n columns on GE TermiNet terminals *only*. Of particular importance is the value -8 , which represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the `nroff(1)` `-h` option for high-speed output. Another special case is the value -0 , implying no tabs at all.
- $n1, n2, \dots$ The arbitrary format lets you type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number except the first one is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1, 10, 20, 30 and 1, 10, +10, +10 are considered identical.
- - *file* If the name of a file is given, `tabs` reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops accordingly; otherwise, it sets them as -8 . This type of specification may be used to ensure that a tabbed file is printed with correct tab settings. For example, use it with the `pr(1)` command in this manner:

```
tabs - - file; pr file
```

OTHER OPTIONS

The following options may also be used. If a given flag occurs more than once, the last value given takes effect.

- Ttype** Tabs usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed under `term(5)`. If no **-T** flag is supplied, tabs searches for the `$TERM` value in the environment. Refer to `environ(5)` for more information on this. If no *type* is found, tabs tries a sequence that works for many terminals.
- +mn** The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a GE TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

CAUTIONS

For some tab settings, GE TermiNet terminals behave differently than most other terminals. The first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type.

Methods to clear tabs and set the left margin are not consistent among various terminals. Usually, you cannot usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

DIAGNOSTICS

- | | |
|---------------------|--|
| "illegal tabs" | Arbitrary tabs incorrectly ordered. |
| "illegal increment" | Arbitrary specification contains a zero or missing increment. |
| "unknown tab code" | "Canned" code cannot be found. |
| "can't open" | A -- <i>file</i> option is used, and the file cannot be opened. |
| "file indirection" | A -- <i>file</i> option is used, and the specification in that file points to yet another file. Indirection of this form is not permitted. |

RELATED INFORMATION

`pr(1)`, `environ(5)`, `term(5)`.

NAME

tail – deliver the last part of a file

USAGE

tail [+- [*number*] [*lbc* [*f*]]] [*file*]

DESCRIPTION

Tail copies the named *file* to the standard output beginning at a designated place. If you do not specify a filename, **tail** uses the standard input.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input. If you fail to specify a *number*, **tail** assumes it to be the value 10. *Number* is counted in units of lines, blocks, or characters, according to the appended option *l*, *b*, or *c*. When you not specify units, counting is by lines.

If the *-f* (“follow”) option is used, and the input file is not a pipe, the program does not terminate after the line of the input file has been copied. Instead, it enters an end-less loop, where it sleeps for a second and then attempts to read and copy further records from the input file. Thus, **tail** may be used to monitor the growth of a file being written by some other process.

EXAMPLE

To print the last 10 lines of *file1*, followed by any lines appended to it between the time **tail** is initiated and killed, use the following:

```
tail -f file1
```

CAUTIONS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.

Various kinds of anomalous behavior may happen with character special files.

RELATED INFORMATION

dd(1).

NAME

tar – tape (and general purpose) archiver

USAGE

tar *key* [*blocksize*] [*name*] *files*

DESCRIPTION

Tar is a general-purpose archiving utility that

- bundles multiple *files* into a single archive file,
- restores to the file system any or all of the *files* in this archive,
- optionally changes permissions on the *files*.

While tar is most often used to archive files on magnetic tape, it can also be used any time you need to bundle a number of files into one object for later reconstitution. The *key* is a string of characters containing one function letter and optional function modifiers. It controls what tar does with the *files* to be bundled or unbundled (restored). Note that in both cases (archiving and restoring), use of a directory name as one of the *files* includes all files in that directory and any subdirectories it contains.

FUNCTIONS*Archiving Files*

The following key letters specify tar options associated with creating an archive file. Unless otherwise specified (see OPTIONS below), the archive is written to the device file */dev/mt/0m*.

- c** Create a new archive. Writing begins at the beginning of the file.
- r** Append to an archive file. Writing begins at the end the file. (Not supported by Apollo tape drivers.)
- u** Add the named *files* to the archive only if they are not already there, or are there but have been modified since they were last archived. (Not supported by Apollo tape drivers.)

Restoring Files

The following key letters specify tar options associated with restoration of selected files from a tar archive file. Unless otherwise specified, the archive from which files are restored is the device file */dev/rmt/0m*. Restored files are always written to the current directory.

- t** List the names of all the files in the archive.
- x** Extract the named *files* from the archive. If one of the *files* is a directory, tar extracts the directory and all files and subdirectories it contains. If no *files* are

specified, **tar x** extracts the entire contents of the archive. Note that if several files with the same name are in the archive, the last one overwrites all earlier ones.

OPTIONS

You may append one or more of the following characters to the function letter in the *key*.

0-9d

This option, valid only when writing archives to tape, specifies the tape drive on which the archive is mounted and, optionally, the density (**I**) of the tape as **l** (low), **m** (medium), or **h** high. The default is drive 0, normally */dev/rmt/0m*.

- b** Set blocksize. This option is valid only when writing archives to tape. Use of this option when writing an archive on a disk file may destroy the archive. If the **b** option is present on the **tar** command line, the *key* must be followed by a space and an integer *blocksize* in the range 1-20. This *blocksize* specifies the block length for tape records. The default is 1, the maximum is 20. Any *blocksize* you specify must coincide with the block length in the magtape descriptor file for the device to which **tar** is writing the archive. When **tar** is reading tapes (keyletters **x** or **t**), it determines the *blocksize* automatically. If the **f** option is also present in the *key*, **tar** expects the *name* and *blocksize* arguments to occur in the same order as the **f** and **b** options. (See examples.)
- f** If the **f** option is present on the **tar** command line, the *key* must be followed by a space and the *name* of the archive you want **tar** to read from or write to. If this option is omitted, the default archive is */dev/rmt/0m*. If *name* is given as a dash (**-**), **tar** writes to the standard output or reads from the standard input, as appropriate. If the **b** option is also present in the *key*, **tar** expects the *name* and *blocksize* arguments to occur in the same order as the **f** and **b** options. (See examples.)
- o** Preserve ownership and permissions of restored files.
- l** Complain when a link to the *file* being archived cannot be resolved.
- m** Do not restore the modification times. Modification time of all restored files is set to the time at which they are restored.
- v** Be verbose. This option makes **tar** display the name of each file it archives or restores, preceded by a function letter that indicates what it is doing (or planning to do) with the file. These function letters are **a** for "append to archive" and **x** for "extract from archive." When combined with the **t** function, **v** makes **tar** print out a "long" listing for each file in the archive, rather than just the name.
- w** This option makes **tar** display the action (see **v** above) it is about to take and the name of the file that will be affected, then wait for confirmation. If you enter a

word beginning with y, tar performs the specified action. Any other input means "no".

EXAMPLES

tar cf /dev/rmt/0m /usr/you

Writes the contents of */usr/you* and all of its subdirectories to */dev/rmt/0m* (medium density magtape).

tar xf /dev/rmt/0m

Restores all of the files from */dev/rmt/0m* to the filesystem, writing them into the current working directory. The person running tar is made owner of the restored files.

tar xfo /dev/rmt/0m

Restores all of the files from */dev/rmt/0m* to the filesystem, writing them into the current working directory. File ownership is preserved as archived.

tar cf /tarfile /usr/you/src/*.c

Bundles all files in */usr/you/src* whose names end in ".c" into */tarfile*.

NOTES

Since the Apollo magnetic and cartridge tape drivers do not support the **u** and **r** functions, tar can only create new archives (tar c) if you're writing to tape. The block length is fixed at 10240 bytes for the files */dev/rmt8* and */dev/rmt12*, 512 bytes for the files */dev/rct8* and */dev/rct12*, and 5120 bytes for the files in */dev/mt/** and */dev/rmt/**. If you need to change the block length, use the */com/edmtdesc* (edit magtape descriptor) command.

If need to re-tension rewind the tape, use the command */com/rbak* with the **-reten** (re-tension) or **-rewind** (rewind) option.

Tar limits filename length to 100 characters, a limit which may affect recursive descent into deep hierarchies.

FILES

/dev/rmt[8 | /0l]

low density magtape, rewind on file close

/dev/rmt[12 | /0ln]

low density magtape, no rewind on file close

/dev/rmt/0m

medium density magtape, rewind on file close

/dev/rmt/0mn

medium density magtape, no rewind on file close

/dev/rct[8 | /c0d0h]

cartridge tape, rewind on file close

/dev/rct[12 | /c0d0hn]

cartridge tape, no rewind on file close

/dev/rct/rfl0a

floppy file name

*/tmp/tar**

temporary file

DIAGNOSTICS

Tar issues various complaints about bad key characters and tape read/write errors.

RELATED INFORMATION

intro(7), /com/rbak, /com/edmtdesc

NAME

tbl – format tables for **nroff** or **troff**

USAGE

tbl [**-TX**] [*files*]

DESCRIPTION

Tbl is a preprocessor that formats tables for **nroff**(1) or **troff**(1). The input files are copied to the standard output, except for lines between **.TS** and **.TE** command lines. **Tbl** assumes these lines to describe tables, so it reformats them as such (without altering their contents).

A **.TS** is followed by the global formatting features indicated below. If these features are used, they are always terminated with a semicolon (;). The global formatting features are followed by line formatting characters, then by lines containing the actual data for the table, followed finally by **.TE**. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only an underscore (_) or an equal sign (=), a single or double line, respectively, is drawn across the table at that point. If a *single item* in a data line consists of only these same characters, then that item is replaced by a single or double line, respectively.

The **-TX** option forces **tbl** to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no filenames are given as arguments, **tbl** reads the standard input, so it may be used as a filter. When it is used with **eqn**(1) or **neqn**(1), **tbl** should come first to minimize the volume of data passed through pipes.

Note: The version of **tbl** supplied with DOMIAN/IX is identical to the one supplied with UNIX System III.

GLOBAL FORMATTING FEATURES

center	Center the table (default is left-adjust).
expand	Make the table as wide as the current line length.
box	Enclose the table in a box.
doublebox	Enclose the table in a double box.
allbox	Enclose each item of the table in a box.
tab (x)	Use the character <i>x</i> instead of a tab to separate items in a line of input data.

LINE FORMATTING KEYLETTERS

Each format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table.

Each column of each line of the table is described by a single keyletter. This keyletter is optionally followed by specifiers that determine what the font and point size of the corresponding item will be; where vertical bars are to appear between columns; and what the column width and intercolumn spacing will be.

The available keyletters are:

c	Center item within the column.
r	Right-adjust item within the column.
l	Left-adjust item within the column.
n	Numerically adjust an item in the column. Vertically align units positions of numbers.
s	Span previous item on the left into this column.
a	Center longest line in this column and then left-adjust all other lines in this column with respect to that centered line.
^	Span down previous entry in this column.
—	Replace this entry with a horizontal line.
=	Replace this entry with a double horizontal line.

The characters **B** and **I** stand for the bold and italic fonts, respectively; the pipe character (|) indicates a vertical line between columns.

EXAMPLE

If we let a colon (:) represent a tab, then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population

—
Town:Households
:Number:Size
=
Bedminster:789:3.26
```

Bernards Twp.:3087:3.74

Bernardsville:2018:3.30

Bound Brook:3425:3.04

Bridgewater:7897:3.81

Far Hills:240:3.19

.TE

yields:

Household Population		
<i>Town</i>	<i>Households</i>	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

RELATED INFORMATION

eqn(1)

mm(1)

mmt(1)

troff(1)

mm(5)

mv(5)

NAME

tee – pipe fitting

USAGE

tee [-i] [-a] [*file*] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*.

OPTIONS

- i Ignore interrupts.
- a Append the output to the *files*, rather than overwriting them.

NAME

test – condition evaluation command

USAGE

test *expr*
[*expr*]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status. If the value is not true, or if no *expr* arguments are specified, a non-zero (false) exit status is returned.

EXPR STRUCTURE

The following primitives are used to construct *expr*:

- | | |
|-----------------------------|--|
| -r <i>file</i> | True if <i>file</i> exists and is readable. |
| -w <i>file</i> | True if <i>file</i> exists and is writable. |
| -x <i>file</i> | True if <i>file</i> exists and is executable. |
| -f <i>file</i> | True if <i>file</i> exists and is a regular file. |
| -d <i>file</i> | True if <i>file</i> exists and is a directory. |
| -l <i>file</i> | True if <i>file</i> exists and is a soft link. |
| -c <i>file</i> | True if <i>file</i> exists and is a character special file. |
| -b <i>file</i> | True if <i>file</i> exists and is a block special file. |
| -p <i>file</i> | True if <i>file</i> exists and is a named pipe (FIFO). |
| -u <i>file</i> | True if <i>file</i> exists and its set-user-ID bit is set. |
| -g <i>file</i> | True if <i>file</i> exists and its set-group-ID bit is set. |
| -k <i>file</i> | True if <i>file</i> exists and its sticky bit is set. |
| -s <i>file</i> | True if <i>file</i> exists and it has a size greater than zero. |
| -t [<i>fildev</i>] | True if the open file whose file descriptor number is <i>fildev</i> (1 by default) is associated with a terminal device. |
| -z <i>s1</i> | True if the length of string <i>s1</i> is zero. |
| -n <i>s1</i> | True if the length of the string <i>s1</i> is non-zero. |
| <i>s1</i> = <i>s2</i> | True if strings <i>s1</i> and <i>s2</i> are identical. |
| <i>s1</i> != <i>s2</i> | True if strings <i>s1</i> and <i>s2</i> are not identical. |

- sl* True if *sl* is *not* the null string.
- n1 -eq n2* True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons *-ne*, *-gt*, *-ge*, *-lt*, and *-le* may be used in place of *-eq*.

These primitives may be combined with the following operators:

- !** Unary negation operator.
- a** Binary *and* operator.
- o** Binary *or* operator (*-a* has higher precedence than *-o*).
- (expr)** Parentheses for grouping.

Notice that all the operators and flags are separate arguments to test. Notice also that parentheses are meaningful to the Shell and, therefore, must be escaped.

CAUTIONS

In the second form of the command, [*expr*], the square brackets must be delimited by blanks. Although DOMAIN/IX recognizes this form of the command, others may not.

RELATED INFORMATION

find(1), sh(1).

NAME

time – time a command

USAGE

time *command*

DESCRIPTION

Time prints the amount of time it took for you to input the *command*, as well as the amount of time it took to execute it. Times are reported in seconds, and are printed on standard error.

EXAMPLE

Time is helpful in reporting the CPU usage of a program. The information is printed on standard error, so commands that typically generate large quantities of output can be timed by sending the standard output to */dev/null*, as in the following example:

```
# time grep e /usr/bin/words > dev/null
```

```
real    7.5
user    2.9
```

The numbers in the output of **time** are elapsed clock (real) time and time spent in the program (user).

RELATED INFORMATION

times(2).

NAME

toc – graphical table of contents routines

USAGE

dtoc [directory]
 ttoc mm-file
 vtoc [-cdhnmvs] [TTOC file]

DESCRIPTION

All of the commands listed below reside in */usr/bin/graf* (see *graphics(1G)*).

dtoc Dtoc makes a textual table of contents, TTOC, of all subdirectories beginning at *directory* (*directory* defaults to *.*). The list has one entry per directory. The entry fields from left to right are level number, directory name, and the number of ordinary readable files in the directory. *Dtoc* is useful in making a visual display of all or parts of a file system. The following will make a visual display of all the readable directories under */*:

dtoc / | vtoc | td

ttoc Output is the table of contents generated by the *.TC* macro of *mm(1)* translated to TTOC format. The input is assumed to be an *mm* file that uses the *.H* family of macros for section headers. If no *file* is given, the standard input is assumed.

vtoc Vtoc produces a GPS describing a hierarchy chart from a TTOC/*fr*. The output drawing consists of boxes containing text connected in a tree structure. If no *file* is given, the standard input is assumed. Each TTOC entry describes one box and has the form:

id [*line-weight,line-style*] "*text*" [*mark*]

where:

id is an alternating sequence of numbers and dots. The *id* specifies the position of the entry in the hierarchy. The *id* 0. is the root of the tree.

line-weight is either:

n, normal-weight; or
m, medium-weight; or
b, bold-weight.

line-style is either:

so, solid-line;
do, dotted-line;
dd, dot-dash line;
da, dashed-line; or

ld, long-dashed

- text* is a character string surrounded by quotes. The characters between the quotes become the contents of the box. To include a quote within a box it must be escaped (\").
- mark* is a character string (surrounded by quotes if it contains spaces), with included dots being escaped. The string is put above the top right corner of the box. To include either a quote or a dot within a *mark* it must be escaped.

Entry example: 1.1 b,da "ABC" DEF

Entries may span more than one line by escaping the new-line (\new-line).

Comments are surrounded by the /*,*/ pair. They may appear anywhere in a TTOC.

Options:

- c** Use text as entered (default is all upper case).
- d** Connect the boxes with diagonal lines.
- hn** Horizontal interbox space is *n* of box width.
- i** Suppress the box *id*.
- m** Suppress the box *mark*.
- s** Do not compact boxes horizontally.
- vn** Vertical interbox space is *n* of box height.

RELATED INFORMATION

graphics(1G).

gps(4) in the *UNIX System Programmer Reference Manual*.

NAME

touch – update access and modification times of a file

USAGE

touch [**-amc**] [*mmddhhmm* [*yy*]] *files*

DESCRIPTION

Touch updates the access and modification times of each of the named files. *File* is created if it does not exist. If no time is specified, the current time is used.

The return code from **touch** is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

OPTIONS

- a** Update access time only.
- m** Update modification time only.
- c** Silently prevent the creation of *file* if it did not previously exist.

RELATED INFORMATION

date(1), **utime(2)**.

NAME

tplot – graphics filters

USAGE

tplot [-T*terminal* [-e *raster*]]

DESCRIPTION

These commands read plotting instructions (see *plot(4)*) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter *\$TERM* (see *environ(5)*) is used. Known *terminals* are:

300 DASI 300.

300S DASI 300s.

450 DASI 450.

4014 TEKTRONIX 4014.

ver Versatec D1200A. This version of *tplot* places a scan-converted image in */usr/tmp/raster\$\$* and sends the result directly to the plotter device, rather than to the standard output. The *-e* option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300

/usr/lib/t300s

/usr/lib/t450

/usr/lib/t4014

/usr/lib/vplot

/usr/tmp/raster\$\$

RELATED INFORMATION

plot(3X), *plot(4)*, *term(5)* in the *UNIX System Programmer Reference Manual*.

NAME

tput – query terminfo database

USAGE

tput [**-T** *type*] *capname*

DESCRIPTION

The **tput** command uses the **terminfo** (4) database to make terminal-dependent capabilities and information available to the Shell. If the attribute *cap(ability) name* is of type *string*, **tput** outputs a string. If it is of type *integer*, **tput** outputs an integer. If the attribute is of type *boolean*, **tput** simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

OPTIONS

-T*type* Specify the terminal *type*. Normally this flag is unnecessary, as the default is taken from the TERM environment variable.

capname Indicate the attribute from the **terminfo**(4) database.

EXAMPLES

tput clear	<i>Echo clear-screen sequence for the current terminal.</i>
tput cols	<i>Print the number of columns for the current terminal.</i>
tput -TVT100 cols	<i>Print the number of columns for the VT100 terminal.</i>
tput hc	<i>Set exit code to indicate if current terminal is a hardcopy terminal.</i>

FILES

<i>/usr/lib/terminfo/?/*</i>	terminal descriptor files
<i>/usr/include/term.h</i>	definition files
<i>/usr/include/curses.h</i>	

DIAGNOSTICS

The **tput** command returns the following error codes:

- 1: Usage error, or terminal has no value for the specified *capname*.
- 2: Bad terminal *type*.
- 3: Bad *capname*.

RELATED INFORMATION

stty(1)
terminfo(4)

NAME

tr – translate characters

USAGE

tr [**-cds**] [*string1* [*string2*]]

DESCRIPTION

Tr copies the standard input to the standard output after substituting or deleting selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options listed below may be used.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

- [*a-z*] Stands for the string of characters with ASCII codes running from character *a* to character *z* inclusive.
- [*a*n*] Stands for *n* repetitions of *a*. If the first digit of *n* is 0, *n* is considered octal; otherwise, *n* is considered decimal. A zero or missing *n* is interpreted as huge; this facility is useful for padding *string2*.

The backslash escape character (\) may be used as it is in the Shell to remove special meaning from any character in a string. In addition, a backslash followed by 1, 2, or 3 octal digits represents the character whose ASCII code is given by those digits.

OPTIONS

- c** Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
- d** Delete all input characters in *string1*.
- s** Squeeze all strings of repeated output characters in *string2* to single characters.

EXAMPLE

The following example creates a list of all the words in *file1* and places them one per line in *file2*, where a word is taken to be a maximal string of alphabetic. The strings are quoted to protect the special characters from interpretation by the Shell; 012 is the ASCII code for newline.

```
# tr -cs "[A-Z][a-z]" "[ 12*]" <file1
```


CAUTIONS

Tr cannot handle ASCII NUL in *string1* or *string2*. It always deletes NUL from input.

RELATED INFORMATION

ed(1), sh(1), ascii(5).

NAME

troff – typeset text

USAGE

troff [*options*] [*files*]

DESCRIPTION

Troff formats text found in *files* (standard input by default) for printing on a phototypesetter.

An argument consisting of a dash (-) is interpreted as a filename corresponding to the standard input.

The options below may appear in any order, but must appear before the *files*.

OPTIONS

- olist Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N-M* means pages *N* through *M*; an initial *-N* means from the beginning to page *N*; and a final *N-* means from *N* to the end.
- nN Number first generated page *N*.
- raN Set register *a* (which must have a one-character name) to *N*.
- i Read standard input after *files* are exhausted.
- q Invoke the simultaneous input/output mode of the *.rd* request.
- z Print only messages generated by *.tm* (terminal message) requests.
- mname Prepend to the input *files* the non-compacted (ASCII text) macro file */usr/lib/tmac/tmac.name*.
- cname Prepend to the input *files* the */usr/lib/macros/cmp.[nt].[dt].name* and */usr/lib/macros/ucmp.[nt].name* compacted macro files.
- kname Compact the macros used in this invocation, placing the output in *[dt].name* files in the current directory.
- sN Stop the phototypesetter every *N* pages (the default is one), produce a trailer to allow changing of cassettes, and resume when the typesetter's start button is pressed. When troff halts between pages, the message *page stop* is sent to the terminal.
- t Direct output to the standard output instead of the phototypesetter.
- f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

- w** Wait until phototypesetter is available, if it is currently busy.
- b** Report whether the phototypesetter is busy or available. No text processing is done.
- a** Send a printable ASCII approximation of the results to the standard output.
- pN** Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- Tname** Use font-width tables for device *name* (the font tables are found in */usr/lib/font/name /**). Currently, no *names* are supported.

CAUTIONS

Troff believes in Eastern Standard Time. As a result, depending on the time of the year and on your local time zone, the date that troff generates may be off by one day from your idea of what the date is.

Using troff with the **-olist** option inside a pipeline, may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

FILES

<i>/usr/lib/suftab</i>	suffix hyphenation tables
<i>/tmp/ta\$#</i>	temporary file
<i>/usr/lib/tmac/tmac.*</i>	standard macro files and pointers
<i>/usr/lib/macros/*</i>	standard macro files
<i>/usr/lib/font/*</i>	font width tables for troff

RELATED INFORMATION

col(1), *cw(1)*, *eqn(1)*, *mm(1)*, *mmt(1)*, *nroff(1)*, *tbl(1)*.

NAME

true, false – provide truth values

USAGE

true

false

DESCRIPTION

True does nothing, successfully. False does nothing, unsuccessfully. Both are typically used in input to `sh(1)`, as in the following example:

```
while true
do
    command
done
```

DIAGNOSTICS

True has an exit status of zero. False has an exit status of non-zero.

RELATED INFORMATION

`sh(1)`.

NAME

tsort – topological sort

USAGE

tsort [*file*]

DESCRIPTION

Tsort produces, on the standard output, a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

CAUTIONS

Tsort uses a quadratic algorithm. This is not worth fixing for the typical use of ordering a library archive file.

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

RELATED INFORMATION

lorder(1).

NAME

tty – get the name of the terminal

USAGE

tty [-l] [-s]

DESCRIPTION

Tty prints the pathname of your terminal.

OPTIONS

- | | |
|----|--|
| -l | Print the synchronous line number connecting your terminal, provided it is an active synchronous line. |
| -s | Inhibit printing of the terminal pathname, allowing testing of the exit code only. |

EXIT CODES

- | | |
|---|-----------------------------------|
| 2 | if invalid options were specified |
| 0 | if standard input is a terminal |
| 1 | otherwise |

DIAGNOSTICS

“not on an active synchronous line”: standard input is not a synchronous terminal and -l is specified.

“not a tty”: standard input is not a terminal and -s is not specified.

NAME

umask – set file-creation mode mask

USAGE

umask [*ooo*]

DESCRIPTION

Umask sets your file-creation mode mask to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. The value of each specified digit is subtracted from the corresponding “digit” specified by the system for the creation of a file. Refer to **creat(2)** for more information about file creation. Also see **chmod(1)** and **umask(2)** for further discussion of file permissions.

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the Shell.

EXAMPLE

To remove write permission of the group and others, execute the following command. Files normally created with mode 777 become mode 755; files created with mode 666 become mode 644.

```
# umask 022
```

RELATED INFORMATION

chmod(1), **sh(1)**, **chmod(2)**, **creat(2)**, **umask(2)**.

NAME

uname – print name of current UNIX system

USAGE

uname [**-snrvma**]

DESCRIPTION

Uname prints the current system name of the UNIX system on the standard output file. It primarily helps to identify which system you are using. The options listed below cause selected information returned by **uname(2)** to be printed.

The **uucp(1)** command looks in the */etc/net/uname* file to determine the site name for the local system. If there is no such file, it assumes *apollo1*.

OPTIONS

- | | |
|-----------|--|
| -s | Print the system name (this is the default). |
| -n | Print the node name (this may be a name by which the system is known to a communications network). |
| -r | Print the operating system release number. |
| -v | Print the operating system version number. |
| -m | Print the machine hardware name. |
| -a | Print all of the above information. |

RELATED INFORMATION

uucp(1), **uname(2)**.

NAME

unget – undo a previous **get** of an SCCS file

USAGE

unget [*-rSID*] [*-s*] [*-n*] *files*

DESCRIPTION

Unget undoes the effect of executing a **get(1)** command with the **-e** option prior to creating the intended new delta.

If you specify a directory as an argument, **unget** behaves as though each file in the directory was specified as a named file, except that it silently ignores non-SCCS and unreadable files.

If you supply a dash (**-**) in place of a file or directory, **unget** reads the standard input, interpreting each line as the name of an SCCS file to be processed.

The options below apply independently to each named file.

OPTIONS

- | | |
|--------------|---|
| -rSID | Uniquely identify which delta is no longer intended, i.e., specified by get(1) as the new delta. This is necessary only if two or more outstanding executions of get(1) for editing on the same SCCS file were done under the same log-in name. A diagnostic results if the specified <i>SID</i> (SCCS IDentification number) is ambiguous, or if it is required but was omitted on the command line. |
| -s | Suppress the printout, on the standard output, of the intended delta's <i>SID</i> . |
| -n | Retain the file secured through the execution of get(1) . This file is normally removed from the current directory. |

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

delta(1), **get(1)**, **help(1)**, **sact(1)**.

NAME

uniq – report repeated lines in a file

USAGE

uniq [**-udc** [**-n**] [**+n**]] [*input* [*output*]]

DESCRIPTION

Uniq reads the input file, and compares adjacent lines. Normally, it removes the second and succeeding copies of repeated lines, and writes the remainder on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent to be found. See **sort(1)** for further details.

OPTIONS

- n** Ignore the first *n* fields, together with any blanks before each. A field is defined as a string of nonspace, nontab characters separated from its neighbors by tabs and spaces. Fields are skipped before characters.
- +n** Ignore the first *n* characters.
- u** Output only those lines not repeated in the original file.
- d** Write a copy of only the repeated lines.
- c** Supersede the **-u** and **-d** options and generate an output report in default style, but with each line preceded by a count of the number of times it occurred.

RELATED INFORMATION

comm(1), **sort(1)**.

NAME

units – conversion program

USAGE

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively, prompting you for information.

Units specifies a quantity as a multiplicative combination of units optionally preceded by a numeric multiplier. It indicates powers by suffixed positive integers, and division by the usual sign.

Units does only multiplicative scale changes; it can convert Kelvin to Rankine, but cannot convert Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, along with some others including the following:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

Pound is not recognized as a unit of mass; lb is. Compound names are run together, (e.g., lightyear). British units that differ from their U.S. counterparts are prefixed like this: brgallon.

EXAMPLES

You have: inch
You want: cm
* 2.54000e+01
/ 3.93701e+00

You have: feet
You want: mile
* 0.189394e-03
/ 0.528000e+04

FILES

/usr/lib/unittab complete list of units

NAME

uucp, uulog, uuname – UNIX system to UNIX system copy

USAGE

uucp [*options*] *source-files destination-file*

uulog [*options*]

uuname [*options*]

DESCRIPTION

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on a machine, or may have the following form:

system name!pathname

where *system name* is taken from a list of system names that **uucp** knows about. Note that only the first six characters of a *system name* are significant. Any excess characters are ignored. The *system name* may also be a list of names such as:

system name!system name!...!system name!pathname

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR. However, you must first insure that intermediate nodes in the route are willing to forward information.

The question mark (?), asterisk (*), and bracketed ellipsis ([...]) Shell metacharacters appearing in *pathname* are expanded on the appropriate system.

Pathnames may be one of the following (anything else is prefixed by the current directory):

a full pathname

a pathname preceded by *~user* where *user* is a log-in name on the specified system and is replaced by that user's log-in directory

a pathname preceded by *~user* where *user* is a log-in name on the specified system and is replaced by that user's directory under PUBDIR

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used. If a simple *~user* destination is inaccessible to **uucp**, data is copied to a spool directory and you are notified by mail(1).

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions. See **chmod(2)** for more information about permissions. All files received by **uucp** will be owned by **uucp**.

Uulog queries a summary log of **uucp** and **uux(1)** transactions in the file */usr/spool/uucp/LOGFILE*. Specific logging information can be obtained by specifying *options*.

Uuname lists the **uucp** names of known systems. A description is printed for each system that has a line of information in */usr/lib/uucp/ADMIN*. The format of **ADMIN** is *sysname tab description tab*.

The DOMAIN/IX version of **uucp** supports the Vadic Autodialer.

OPTIONS

The following options are interpreted by **uucp only** :

- d** Make all necessary directories for the file copy (default).
- c** Use the source file when copying out rather than copying the file to the spool directory (default).
- mfile** Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed. This option only works for sending files or receiving a single file. Receiving multiple files specified by a question mark (?), asterisk (*), or bracketed ellipsis ([...]) special Shell character will not activate the option. This option will not work if all transactions are local or if **uucp** is executed remotely via the **-e** option.
- r** Queue the job but do not start the file transfer process. By default a file transfer process is started each time **uucp** is evoked.
- sdirectory** Spool **uucp** files in *directory*.
- gn** Give the named **uucp** file a grade of *n*, which affects when the file is processed.
- xlevel** Provide *level* of debugging information. *Level* should be a numeric value, and the greater the value used, the more debug information that is produced.

The following options are used by **uulog only**:

- ssys** Print information about work involving system *sys*. If *sys* is not specified, print logging information for all systems.
- uuser** Print information about work done for the specified *user*. If *user* is not specified, print logging information for all users.

The following option is used by **uname** *only*:

-l Return the local system name.

CAUTIONS

The domain of remotely accessible files may (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path-name. Ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin with */usr/spool/uucppublic* (equivalent to *~nuucp* or just *~*).

To send files that begin with a period (e.g., *.profile*), you must qualify the files with a period. For example: *.profile*, *.prof**, and *.profil?* are correct; whereas **prof** and *?profile* are incorrect.

FILES

<i>/usr/spool/uucp</i>	spool directory
<i>/usr/spool/uucppublic</i>	public directory for receiving and sending (PUBDIR)
<i>/usr/lib/uucp/*</i>	other data and program files

Uucp looks in */etc/net/uname* to determine the site name for the local system. If there is no such file, it assumes *apollo1*.

RELATED INFORMATION

mail(1), **uux(1C)**, **chmod(2)**.

Also refer to the discussion of **uucp** in the *Domain/IX User's Guide*.

NAME

uustat – **uucp** status inquiry and job control

USAGE

uustat [*options*]

DESCRIPTION

Uustat can display the status of, or cancel, previously specified **uucp(1C)** commands. It can also provide general status on **uucp(1C)** connections to other systems.

When no options are given, **uustat** outputs the status of all **uucp** requests that you have issued. Note that only one of the options **-j**, **-m**, **-k**, **-c**, **-r**, can be used with the rest of the other options.

The meanings of the job request status are as follows, where *status* may be either an octal number or a verbose description:

job-number user remote-system command-time status-time status

The octal code corresponds to the following descriptions:

OCTAL	STATUS
000001	copy failed for undetermined reason
000002	permission to access local file denied
000004	permission to access remote file denied
000010	bad uucp(1C) command generated
000020	remote system cannot create temporary file
000040	cannot copy to remote directory
000100	cannot copy to local directory
000200	local system cannot create temporary file
000400	cannot execute uucp(1C)
001000	copy (partially) succeeded
002000	copy finished, job deleted
004000	job queued
010000	job killed (incomplete)
020000	job killed (complete)

The meanings of the machine accessibility status are as follows, where *time* is the latest status time and *status* is a self-explanatory description of the machine status:

system-name time status

OPTIONS

-j*jobn* Report the status of the **uucp(1C)** request *jobn*. If *all* is used for *jobn*, report the status of all **uucp(1C)** requests. An argument must be supplied. Otherwise, the usage message is printed and the request fails.

- kjobn** Kill the **uucp** request whose job number is *jobn*. The killed request must belong to the person issuing the **uustat** command (unless this is being done by the super-user).
- rjobn** Rejuvenate *jobn*, setting its modification time to the current time.
- chour** Remove the status entries older than *hour* hours. This can only be initiated by the user **uucp** or the super-user.
- uuser** Report the status of all **uucp**(1C) requests issued by *user*.
- ssys** Report the status of all **uucp**(1C) requests communicating with remote system *sys*.
- ohour** Report the status of all **uucp** requests older than *hour* hours.
- yhour** Report the status of all **uucp**(1C) requests younger than *hour* hours.
- mmch** Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, provide the status of all machines known to the local **uucp**(1) capability.
- Mmch** Perform the same activity as the **-m** option, except print both the time of the last status report and the time of the last successful transfer to the associated system.
- Ojobn** Report the **uucp**(1C) status of *jobn*, using the octal status codes listed on the previous page. If *jobn* is specified as **all**, report the status of all jobs in octal. If neither form of the **-O** option is not specified, a verbose description is printed with each **uucp** request.
- q** List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, list its date of creation.

EXAMPLE

To print the status of all **uucp** requests issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours, execute the following:

```
# uustat -uhdc -smhtsa -y72
```

FILES

<i>/usr/spool/uucp</i>	spool directory
<i>/usr/lib/uucp/L_stat</i>	system status file
<i>/usr/lib/uucp/R_stat</i>	request status file

UUSTAT (1C)

DOMAIN/LX SYS5

UUSTAT (1C)

RELATED INFORMATION

uucp(1C).

NAME

uuto, **uupick** – public UNIX-to-UNIX system file copy

USAGE

uuto [*options*] *source-files destination*

uupick [-s *system*]

DESCRIPTION

Uuto sends *source-files* to *destination*. It uses the **uucp**(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a pathname on your machine. *Destination* has this form:

system!user

where *system* is taken from a list of system names that **uucp**(1C) knows about. Refer to

uname(1) for more information. The *user* is the log-in name of someone on the specified system.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the **uucp**(1C) source. Specifically the files are sent to this address:

PUBDIR/receive/*user*/*mysystem*/files

The destined recipient is notified by **mail**(1) when the files arrive.

Uupick accepts or rejects the files transmitted to you. Specifically, **uupick** searches PUBDIR for files destined for you. For each entry (file or directory) found, **uupick** prints the following message on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

<newline>	Continue to next entry.
d	Delete the entry.
m [<i>dir</i>]	Move the entry to named directory <i>dir</i> (the current directory is the default).
a [<i>dir</i>]	Same as m except moving all the files sent from <i>system</i> .
p	Print the content of the file.
q	Stop.

EOT (↑Q)	Same as q.
!command	Escape to the Shell to do <i>command</i> .
*	Print a command summary.

OPTIONS

-p	Copy the source file into the spool directory before transmission. For use with uuto <i>only</i> .
-m	Send mail to the sender when the copy is complete. For use with uuto <i>only</i> .
-s <i>system</i>	Only search the PUBDIR for files sent from <i>system</i> . For use with uupick <i>only</i> .

CAUTIONS

To send files that begin with a period (e.g., *.profile*), you must qualify the files with a period. For example: *.profile*, *.prof**, and *.profil?* are correct; whereas **prof** and *?profile* are incorrect.

FILES

PUBDIR/usr/spool/uucppublic public directory

RELATED INFORMATION

mail(1), uucp(1C), uustat(1C), uux(1C).

NAME

uux – UNIX-to-UNIX system command execution

USAGE

uux [*options*] *command-string*

DESCRIPTION

Uux gathers zero or more files from various systems, executes a command on a specified system, and then sends standard output to a file on a specified system. Note that, for security reasons, many installations limit the list of commands executable on behalf of an incoming request from **uux**. Many sites permit little more than the receipt of **mail(1)** via **uux**.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that *system name!* may prefix the command and filenames. A null *system name* is interpreted as the local system.

Filenames may be one of the following:

- a full pathname

- a pathname preceded by *~xxx* where *xxx* is a log-in name on the specified system and is replaced by that user's log-in directory

- anything else is prefixed by the current directory

Any special Shell characters such as *<>|* should be quoted, either by quoting the entire *command-string*, or by quoting the special characters as individual arguments.

Uux attempts to get all files to the execution system. For output files, the filename must be escaped using parentheses. **Uux** notifies you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine. Executable commands are listed in */usr/lib/uucp/L.cmds* on the remote system. The format of the *L.cmds* file is:

cmd,machine1,machine2,...

If no machines are specified, any machine can execute *cmd*. If machines are specified, only the listed machines can execute *cmd*. If the desired command is not listed in *L.cmds*, then no machine can execute that command.

Redirection of standard input and output is usually restricted to files in **PUBDIR**. Directories into which redirection is allowed must be specified in */usr/lib/uucp/USERFILE* by the system administrator.

The environment variable `JOBNO` and the `-j` option are used to control the listing of the `uux` job number on standard output. If the environment variable `JOBNO` is undefined or set to `OFF`, the job number is not listed (this is the default). If `uux` is then invoked with the `-j` option, the job number is listed. If the environment variable `JOBNO` is set to `ON` and is exported, a job number is written to standard output each time `uux` is invoked. In this case, the `-j` option suppresses output of the job number.

OPTIONS

- Make the standard input to `uux` the standard input to the *command-string*.
- n Send no notification to user.
- m*file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.
- j Control writing of the `uucp(1C)` job number to standard output. `Uux` associates a job number with each request. This job number can be used by `uustat(1C)` to obtain status or terminate the job.
- x*number*
Print debugging information up to degree *number*.

EXAMPLES

To get the *fl* files from the “*usg*” and “*pwba*” machines, execute a `diff(1)` command, and put the results in *fl.diff* in the local directory:

```
# uux “!diff usg!/usr/dan/fl pwba!/a4/dan/fl >
```

To send a

`uucp(1C)` command to system “*a*” to get `/usr/file` from system “*b*” and send it to system “*c*”:

```
# uux a!uucp b!/usr/file \c!/usr/file\
```

CAUTIONS

Only the first command of a shell pipeline may have a *system name!* in it. All other commands are executed on the system of the first command.

The use of the asterisk (*) shell metacharacter will probably not do what you want it to do.

The shell tokens `<<` and `>>` are not implemented.

Only the first six characters of the *system name* are significant. `Uux` ignores any excess characters.

FILES

<code>/usr/spool/uucp</code>	spool directory
<code>/usr/spool/uucppublic</code>	public directory (PUBDIR)
<code>/usr/lib/uucp/*</code>	other data and programs

RELATED INFORMATION

mail(1), uucp(1C).

Also refer to the discussion of **uucp** in the *Domain/IX System Administrator's Reference*.

NAME

val – validate SCCS file

USAGE

val –
val [-s] [-rSID] [-mname] [-ytype] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file that meets the characteristics specified by the optional argument list. Options may appear in any order, as long as they are listed before filenames. The effects of any option applies independently to each named file on the command line.

Val uses the dash (–) as a special argument, causing the standard input to be read until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The 8-bit code returned by val is a disjunction of the possible errors, i.e., it can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument
- bit 1 = unknown or duplicate keyletter argument
- bit 2 = corrupted SCCS file
- bit 3 = cannot open file or file not SCCS
- bit 4 = SID is invalid or ambiguous
- bit 5 = SID does not exist
- bit 6 = Y, -y mismatch;
- bit 7 = M, -m mismatch;

Note that val can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases, an aggregate code is returned – a logical OR of the codes generated for each command line and file processed.

OPTIONS

- | | |
|-------|---|
| -s | Silence the diagnostic message normally generated on the standard output for any error detected while processing each named file on a given command line. |
| -rSID | Check to see whether the value of <i>SID</i> (an SCCS delta number) is ambiguous or invalid. For example, r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist. Furthermore, r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta |

number. If the SID is valid and not ambiguous, this option also checks to see if it actually exists.

- mname** Compare the value of *name* with the SCCS M keyword in *file*.
- ytype** Compare the value of *type* with the SCCS Y keyword in *file*.

CAUTIONS

Val can process up to 50 files on a single command line.

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

admin(1), **delta(1)**, **get(1)**, **help(1)**, **prs(1)**.

NAME

vc – version control

USAGE

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

DESCRIPTION

Vc copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user-declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the -t option (see below). The default control character is a colon (:), except as modified by the -c option (see below). Input lines beginning with a backslash (\) and followed by a control character are not control lines. These lines are copied to the standard output with the backslash removed. Lines beginning with a backslash and followed by a non-control character are copied in their entirety.

A keyword is composed of nine or fewer alphanumeric; the first must be alphabetic. A value is any ASCII string that can be created with ed(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Vc replaces keywords by values whenever it finds a keyword surrounded by control characters on a version control statement. The -a option (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with a backslash. If a literal \ is desired, then it too must be preceded by a backslash.

OPTIONS

- | | |
|--------|---|
| -a | Force replacement of keywords surrounded by control characters with their assigned value in all text lines. |
| -t | Ignore all characters from the beginning of a line up to and including the first <i>tab</i> character, for the purpose of detecting a control statement. If one is found, discard all characters up to and including the <i>tab</i> . |
| -cchar | Specify a control character to be used in place of a colon. |
| -s | Silence warning messages (not error) that are normally printed on the diagnostic output. |

VERSION CONTROL STATEMENTS

:del *keyword* [, ..., *keyword*]

Declare *keywords*. All keywords must be declared.

:asg *keyword*=*value*

Assign *values* to *keywords*. An asg statement overrides the assignment for the corresponding keyword on the vc command line and all previous asg's for that keyword. Keywords declared but not assigned values have null values.

:if *condition*

⋮

:end

Skip lines of the standard input. If the condition is true, all lines between the if statement and the matching end statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening if statements and matching end statements are recognized solely for the purpose of maintaining the proper if-end matching.

The syntax of a condition is:

<cond>	::= ["not"] <or>
<or>	::= <and> <and> " " <or>
<and>	::= <exp> <exp> "&" <and>
<exp>	::= "(" <or> ")" <value> <op> <value>
<op>	::= "=" "!=" "<" ">"
<value>	::= <arbitrary ASCII string> <numeric string>

The available operators and their meanings are:

=	equal
!=	not equal
&	and
	or
>	greater than
<	less than
()	used for logical groupings
not	may only occur immediately after the if, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence
&
|

Values must be separated from operators or parentheses by at least one blank or tab.

Parentheses may be used to alter the order of precedence.

::text

Replace keywords on lines copied to the standard output. Remove the two leading control characters, and replace keywords surrounded by control characters in *text* by their value before copying the line to the output file. This action is independent of the *-a* option.

:on

:off

Turn on or turn off keyword replacement on all lines.

:ctl *char*

Change the control character to *char*.

:msg *message*

Print the given *message* on the diagnostic output.

:err *message*

Print the given *message* followed by:

ERROR: err statement on line ... (915)

on the diagnostic output. Halt vc execution, and return an exit code of 1.

EXIT CODES

0 – normal

1 – any error

DIAGNOSTICS

Use **help(1)** for explanations.

RELATED INFORMATION

ed(1), **help(1)**.

NAME

vi – full screen display editor based on **ex**

USAGE

vi [-t *tag*][-r *crashfile*][-l][-wn][-R][+*command*] *file(s)*
view [-t *tag*][-r *crashfile*][-l][-wn][+*command*] *file(s)*
vedit [-t *tag*][-r *crashfile*][-l][-wn][-R][+*command*] *file(s)*

DESCRIPTION

Vi (visual) is a display-oriented text editor based on the line editor **ex**(1). You can use the command mode of **ex** from within **vi** and vice-versa.

When using **vi**, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates your position within the file. The *DOMAIN/IX Text Editors Quick Reference* card, the *Introduction to Display Editing with Vi* and the *Ex Reference Manual*, all of which are part of the *DOMAIN/IX Text Processing Guide*, provide full details on using **vi**.

OPTIONS

When you invoke **vi** with no options, as

vi file

it opens *file* and positions the cursor at the first line. To quit **vi**, type:

:q

If you have modified *file*, **vi** will print a “no write since last change” message. The command

:wq

writes the changed version of *file*, then quits **vi**, while the command

:q!

forces the quit without a write.

Vi recognizes the following command line options:

- | | |
|---------------------|---|
| -t tag | Edit the file containing the <i>tag</i> and position the editor at its definition. |
| -r crashfile | Recover <i>crashfile</i> after an editor or system crash. If <i>crashfile</i> is not specified a list of all saved files will be printed. |

- l LISP mode; indents appropriately for LISP code, the () {} [[and]] commands in vi and open are modified to have meaning for LISP
- wn Set the default window size to *n*. This is useful when running the editor over a slow speed line.
- R Read only mode; the readonly flag is set, preventing accidental overwriting of the file.
- +command The specified ex command is interpreted before editing begins.
- file(s) one or more files to be edited.

The view invocation is the same as vi except that the readonly flag is set.

The vedit invocation is intended for beginners. In vedit, the report flag is set to 1, and the showmode and novice flags are set. These defaults make it easier to get started learning the editor.

VI MODES

- Command Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
- Input Entered by a i A I o O c C s S R, after which arbitrary text may be entered. Input mode is normally terminated with an ESC character, or abnormally with interrupt.
- Last line Reading input for : / ? or !; terminate with RETURN to execute or INTerrupt to cancel.

COMMAND SUMMARY

Sample Commands

- | | |
|----------|----------------------------|
| ← ↓ ↑ → | arrow keys move the cursor |
| h j k l | same as arrow keys |
| itextESC | insert <i>text</i> |
| cwnewESC | change word to <i>new</i> |
| eaESC | pluralize word |
| x | delete a character |
| dw | delete a word |
| dd | delete a line |
| 3dd | ... 3 lines |
| u | undo previous change |
| ZZ | exit vi, saving changes |
| :q!CR | quit, discarding changes |
| /textCR | search for <i>text</i> |
| ^U ^D | scroll up or down |

`:ex cmdCR` any ex or ed command

Counts Before vi Commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	<code>z G </code>
scroll amount	<code>^D ^U</code>
repeat effect	most of the rest

Interrupting and Canceling

ESC	end insert or incomplete cmd
<code>^?</code>	(delete or rubout) interrupts
<code>^L</code>	reprint screen if <code>^?</code> scrambles it
<code>^R</code>	reprint screen if <code>^L</code> is <code>→</code> key

File Manipulation

<code>:wCR</code>	write back changes
<code>:qCR</code>	quit
<code>:q!CR</code>	quit, discard changes
<code>:e nameCR</code>	edit file <i>name</i>
<code>:e!CR</code>	reedit, discard changes
<code>:e + nameCR</code>	edit, starting at end
<code>:e +nCR</code>	edit starting at line <i>n</i>
<code>:e #CR</code>	edit alternate file
	synonym for <code>:e #</code>
<code>:w nameCR</code>	write file <i>name</i>
<code>:w! nameCR</code>	overwrite file <i>name</i>
<code>:shCR</code>	shell escape
<code>:!cmdCR</code>	run <i>cmd</i> , then return
<code>:nCR</code>	edit next file in arglist
<code>:n argsCR</code>	specify new arglist
<code>^G</code>	show current file and line
<code>:ta tagCR</code>	to tag file entry <i>tag</i>
<code>^]</code>	<code>:ta</code> , following word is <i>tag</i>

In general, any ex or ed command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a CR.

Positioning Within File

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/pat	next line matching <i>pat</i>
?pat	prev line matching <i>pat</i>
n	repeat last / or ?
N	reverse last / or ?
/pat/+n	<i>n</i> th line after <i>pat</i>
?pat?-n	<i>n</i> th line before <i>pat</i>
]]	next section/function
[[previous section/function
(beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
	find matching () { or }

Adjusting the Screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z-CR	... at bottom
z.CR	... at center
/pat/z-CR	<i>pat</i> line at bottom
zn.CR	use <i>n</i> line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and Returning

``	move cursor to previous context
''	... at first non-white in line
mx	mark current position with letter x
`x	move cursor to mark x
'x	... at first non-white in line

Line Positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character Positioning

^	first non-white
0	beginning of line
\$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
fx	find x forward
Fx	f backward
tx	upto x forward
Tx	back upto x
;	repeat last f F t or T
,	inverse of ;
	to specified column
	find matching ({) or }

Words, Sentences, and Paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

LISP Mode

)	Forward s-expression
}	... but do not stop at atoms
(Back s-expression
{	... but do not stop at atoms

Corrections During Insert

^H	erase last character
^W	erase last word
erase	your erase, same as ^H
kill	your kill, erase input this line
\	quotes ^H, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over <i>autoindent</i>
↑^D	kill <i>autoindent</i> , save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and Replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
rx	replace single char with x
RtextESC	replace characters, one-for-one

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since w moves over a word, dw deletes the word that would be moved over. Double the operator, e.g. dd to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command
=	indent for LISP

Miscellaneous Operations

C	change rest of line (c\$)
D	delete rest of line (d\$)
s	substitute chars (cl)
S	substitute lines (cc)
J	join lines
x	delete characters (dl)
X	... before cursor (dh)
Y	yank lines (yy)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"xp	put from buffer <i>x</i>
"xy	yank to buffer <i>x</i>
"xd	delete into buffer <i>x</i>

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"d p	retrieve <i>d</i> 'th last delete

RELATED INFORMATION

ex(1)

*DOMAIN/IX Text Processing Guide**DOMAIN/IX Text Editors Quick Reference*

NAME

wait – await completion of process

USAGE

wait

DESCRIPTION

Wait does nothing until all processes starting with an ampersand (&) have completed. It then reports abnormal terminations.

Because the wait(2) system call must be executed in the parent process, the shell itself executes wait without creating a new process.

Note: All processes with a three- or more-stage pipeline are not children of the shell. Therefore, all are not awaited.

RELATED INFORMATION

sh(1), wait(2).

NAME

wc – word count

USAGE

wc [**-lwc**] [*files*]

DESCRIPTION

Wc counts lines, words, and characters in a named file, or in the standard input if you do not specify a *file*. A word is defined as a maximal string of characters delimited by spaces, tabs, or newlines. If you give more than one filename as an argument, **wc** performs an individual count on each file, as well as a total count of lines, words, and characters in all files specified. In any case, it always lists the names of specified *files* along with the count.

OPTIONS

You may use the **l**, **w**, and **c** options in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

NAME

what – identify SCCS files

USAGE

what [-s] *files*

DESCRIPTION

What searches the given files for all occurrences of the pattern that **get(1)** substitutes for **Z** (this is **@(#)** at this printing). It then prints what follows until the first tilde (~), greater-than character (>), newline, backslash (\), or null character.

Although **what** is meant to be used with **get(1)**, which automatically inserts identifying information, you can also use it where information is inserted manually.

Specifying the **-s** option causes **what** to quit after finding the first occurrence of a pattern in each file.

EXAMPLE

If the C program in file *f.c* contains

```
static char ident[ ] = "@(#)identification information";,
```

and *f.c* is compiled to yield *f.o* and *a.out*, then

```
# what f.c f.o a.out
```

prints the following information:

```
f.c:          identification information
```

```
f.o:          identification information
```

```
a.out:        identification information
```

CAUTIONS

By chance, an unintended occurrence of the **@(#)** pattern could be found, but this causes no harm in most cases.

DIAGNOSTICS

Exit status is 0 if any matches are found; otherwise, it is 1.

Use **help(1)** for explanations.

WHAT(1)

DOMAIN/LX SYS5

WHAT(1)

RELATED INFORMATION
get(1), help(1).

NAME

who – who is on the system

USAGE

who [-a]
who am I

DESCRIPTION

Who is a program that lists users logged into a single node, or to all nodes on the network. It can also be used to find out who the system thinks you are (if, for example, you run **login(1)** and become someone else).

OPTIONS

When you invoke **who** with no argument, it lists all users logged into your node. This includes the user who most recently logged into the DM, users logged in via **rlogin**, and users logged in via SIO lines.

-a lists the log-in name and node ID of each user currently logged into the Display Manager on each node on the network.

who am I returns your current user id, node name, and node id. The forms “**who am i**” and even “**who are you**” are equivalent.

EXAMPLES

```
who
bob                ttypl    Dec 3 11:48    (vax_1)
```

```
who -a
bob          node1054
john         node2CBC
freddy       node4B7
braf         node5FE
bcbell       node2A64
rm           node706
```

```
who am I
xnode!bob    node1054
```


NOTES

Who treats the DM as a single display, so if you run **login** in a shell managed by the DM, the person you log in as replaces the person who last logged into the DM or any shell window. After you log out of a DM shell window, **who** continues to report that the person you were logged in as is logged into the DM.

NAME

write – write to another user

USAGE

write *node!user* [*ttyname*]
write *user@node* [*ttyname*]

DESCRIPTION

The write command copies lines from your node to that of another user. When first called, it sends the following message:

Message from yournode!yourname [yourttyname] ...

The recipient of the message should write back at this point. Communication continues until an end-of-file (EOF) is read from the terminal, or an interrupt is sent. At that point, the write program writes "EOT" on the other terminal and exits.

To write to a user who is logged in more than once, use the *ttyname* argument to indicate the appropriate terminal name.

If write finds an exclamation point (!) at the beginning of a line, it calls the shell to execute the rest of the line as a command.

When you first write to a user, wait for a reply before sending another message. Each party should end each message with a distinctive signal, such as (o) for "over", or (oo) for "over and out" (when you want to terminate the conversation).

In order to accept incoming write messages, a node must be running the `writed(8c)` process, as well as the `DOMAIN mbx_helper` process. Before SR9.5, it was necessary to start an `mbx_helper` explicitly, but after SR9.5, `writed` automatically starts an `mbx_helper`.

FILES

<i>/etc/utmp</i>	record of who is logged in on the node (link to ' <i>node_data/etc.utmp</i>)
<i>/bin/sh</i>	to execute "!"

RELATED INFORMATION

`who(1)`
`mail(1)`

NAME

xargs – construct argument list(s) and execute a command

USAGE

xargs [*options*] [*command* [*initial arguments*]]

DESCRIPTION

Xargs combines the fixed *initial arguments* with arguments read from standard input to execute the specified *command* one or more times. The options that you specify determine the number of arguments read for each *command* invocation and the manner in which they are combined.

Xargs uses your \$PATH to search for *command*, which may be a Shell file. If you omit *command*, it uses the /bin/echo command.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside quoted strings, a backslash (\) escapes the next character.

Each argument list begins with the *initial arguments*, followed by some number of arguments read from standard input (with the exception of those using the -i option). The -i, -l, and -n options determine how arguments are selected for each command invocation. When none of these options are coded, the *initial arguments* are followed by arguments read continuously from standard input until an internal buffer is full. Then, *command* is executed with the accumulated arguments. Xargs repeats this process until there are no more arguments. When options conflict (e.g., -l vs. -n), the last option has precedence.

OPTIONS

-l*number*

Execute the *command* for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline *unless* the last character of the line is a blank or a tab. A trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, one is assumed. The -x option is forced.

-i*replstr*

Insert mode: execute *command* for each line from standard input, taking the entire line as a single argument, inserting it in *initial arguments* for each occurrence of *replstr*. A maximum of five arguments in *initial arguments* may each contain one or more instances of *replstr*. Discard blanks and tabs at the beginning of each line. Constructed arguments may not grow larger than 255

characters, and option `-x` is also forced. Braces (`{ }`) are assumed for *replstr* if not specified.

- `-nnumber` Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option `-x` is also coded, each *number* arguments must fit in the *size* limitation. Otherwise, *xargs* terminates execution.
- `-t` Trace mode: echo *command* and each constructed argument list to file descriptor 2 just prior to their execution.
- `-p` Prompt mode: ask whether or not to execute *command* each invocation. Turn on trace mode (`-t`) to print the command instance to be executed, followed by a `?... prompt`. Replying with a `y` (optionally followed by anything) executes the command. Anything else, including a simple carriage return, skips that particular invocation of *command*.
- `-x` Terminate *xargs* if any argument list is greater than *size* characters. The `-x` option is forced by the `-i` and `-l` options. When the `-i`, `-l`, or `-n` options are coded, the total length of all arguments must be within the *size* limit.
- `-ssize` Set the maximum total size of each argument list to *size* characters. *Size* must be a positive integer less than or equal to 470. If the `-s` option is not coded, 470 becomes the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
- `-eofstr` Use *eofstr* as the logical end-of-file string. An underscore (`_`) is assumed for the logical EOF string if the `-e` option is not coded. The value of `-e` with no *eofstr* coded turns off the logical EOF string capability (an underscore is taken literally). *Xargs* reads standard input until it encounters either an end-of-file or the logical EOF string.

EXAMPLES

To move all files from directory `$1` to directory `$2`, and to echo each move command just before executing, use the following:

```
ls $1 | xargs -i -t mv $1/{ } $2/{ }
```


To combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*, use this:

```
(logname; date; echo $0 $*) | xargs >>log
```

To execute `diff(1)` with successive pairs of arguments originally typed as Shell arguments:

```
echo $* | xargs -n2 diff
```

CAUTIONS

Xargs terminates if it receives a return code of `-1` from, or if it cannot execute, *command*. When *command* is a Shell program, it should explicitly exit with an appropriate value to avoid accidentally returning with `-1`. Consult `sh(1)` for more information about how the Shell works.

DIAGNOSTICS

Self-explanatory.

RELATED INFORMATION

`sh(1)`.

NAME

yacc – yet another compiler-compiler

USAGE

yacc [-vdlit] *grammar*

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton that executes an LR parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The C compiler must compile the *y.tab.c* output file to produce a *yyparse* program. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error-handling routine. You supply these routines. The *lex(1)* command can help create lexical analyzers for use by yacc.

OPTIONS

- v Prepare the *y.output* file, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.
- d Generate the *y.tab.h* file with the *#define* statements that associate the yacc-assigned token codes with the user-declared token names. This allows source files other than *y.tab.c* to access the token codes.
- l Do not include any *#line* constructs in the code produced in the *y.tab.c* file. This should only be used after the grammar and the associated actions are fully debugged.
- t Compile runtime debugging code by default. Runtime debugging code is always generated in *y.tab.c* under conditional compilation control. Normally, this code is not included when *y.tab.c* is compiled. Regardless of whether or not this option is used, the runtime debugging code is always under the control of *YYDEBUG*, a preprocessor symbol. If *YYDEBUG* has a non-zero value, then the debugging code is included. If its value is zero, the code is not included. The size and execution time of a program produced without the runtime debugging code is smaller and slightly faster.

CAUTIONS

Because filenames are fixed, only one yacc process may be active in a given directory at a time.

FILES

<i>y.output</i>	
<i>y.tab.c</i>	
<i>y.tab.h</i>	defines for token names
<i>yacc.tmp</i>	temporary file
<i>yacc.debug</i>	temporary file
<i>yacc.acts</i>	temporary file
<i>/usr/lib/yaccpar</i>	parser prototype for C programs

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output. (You can find a more detailed report in the *y.output* file.) Similarly, if some rules cannot be reached from the start symbol, this is also reported.

RELATED INFORMATION

lex(1), malloc(3C).

This is a topical index for Section 1 of the *DOMAIN/IX Command Reference Manual for SYS5*. For a permuted index of all reference information, see Appendix A of this manual.

AEGIS Shell	1-103, 1-308
AEGIS commands,	
INLIB	1-81
arrays	1-194, 1-215
ASA carriage control characters	1-9
ASCII character code	1-106, 1-125, 1-149, 1-160, 1-189, 1-270, 1-364
Bourne Shell	1-94, 1-101, 1-103, 1-139, 1-148, 1-152, 1-158, 1-219, 1-223, 1-224, 1-266, 1-306, 1-340, 1-368, 1-371, 1-409
buffers	1-140, 1-215, 1-347
C Shell	1-103, 1-139, 1-152
C programs,	
beautification of	1-39,
checking for errors in	1-46
cross-reference for	1-110
cross-reference for	1-46
debugging	1-95
operators used in	1-63
C shell	1-67
C operators	1-13
calendars	1-36, 1-37
commands,	
getting help on SCCS	1-181
immunity to hangups and quits	1-82, 1-266, 1-317
making argument lists for	1-139, 1-407
on-line documentation of	1-247
parsing options in	1-172, 1-244
timing of	1-82, 1-262, 1-320, 1-358
compilers	1-26, 1-40, 1-61, 1-198, 1-244, 1-295, 1-410
current directory	1-140, 1-247
current line	1-140
daemons	1-406
databases	1-189
dates,	
file access/modification	1-59, 1-161, 1-165, 1-199, 1-212, 1-361
setting of	1-64, 1-112
debuggers	1-26, 1-41, 1-95, 1-108, 1-410
deltas,	
SCCS	1-127, 1-166, 1-283, 1-297, 1-298, 1-373, 1-387
desk calculator	1-122

devices,	1-343
auto-dialer	1-100, 1-377
disks	1-138
printer	1-153, 1-290
tape	1-348
tape drives	1-126
terminal	1-247, 1-251, 1-336, 1-344, 1-370
directories,	
changing	1-43, 1-102, 1-245, 1-306
comparing	1-137
creating	1-59, 1-250
current	1-202, 1-292
home	1-65, 1-67, 1-340
listing	1-137
listing contents of	1-138, 1-212
removing	1-296
renaming	1-255
searching for files in	1-88, 1-132, 1-161, 1-213, 1-313
working	1-43, 1-292
disk usage	1-138
document preparation	1-53, 1-104, 1-106, 1-130, 1-153, 1-182, 1-251, 1-253, 1-256, 1-281, 1-290, 1-325, 1-352
DOMAIN C Language Reference	1-40, 1-61
DOMAIN C compiler	1-41, 1-61, 1-244
DOMAIN C preprocessor	1-46, 1-47, 1-61, 1-199
Display Manager	1-404
EBCDIC character code	1-125
editors,	
line	1-132, 1-140, 1-226, 1-300, 1-302
stream	1-302
visual	1-150, 1-155
environment	1-83, 1-84, 1-152, 1-204, 1-206, 1-223, 1-231, 1-239, 1-247, 1-251, 1-261, 1-306, 1-312, 1-340, 1-385
variables	1-259
expressions,	
arithmetic	1-14, 1-18, 1-76
evaluation of	1-157, 1-356
regular	1-14, 1-22, 1-76, 1-141, 1-177, 1-194, 1-295, 1-302
relational	1-14
file protection	1-259
files,	
archive	1-6, 1-8, 1-59, 1-207, 1-244, 1-369
classifying	1-160

comparing	1-21, 1-52, 1-57, 1-132, 1-134, 1-136, 1-137, 1-299, 1-300, 1-374
compressing/expanding	1-271
concatenating	1-38, 1-275
converting	1-53, 1-125, 1-130, 1-132, 1-256
copying	1-58, 1-59, 1-102, 1-125, 1-202, 1-347
cutting/pasting	1-104, 1-275
deleting	1-296
determining size of	1-138, 1-140, 1-150, 1-212, 1-342, 1-401
dumping	1-270
merging	1-189, 1-275, 1-321
moving	1-255
numbering lines in	1-263
object	1-319
ownership/permission of	1-49, 1-51, 1-59, 1-161, 1-212, 1-250, 1-255, 1-271, 1-297, 1-313, 1-371, 1-377
perusal of	1-277
printing	1-281
removing	1-202
renaming	1-58, 1-202
scanning	1-22, 1-177
sorting	1-57, 1-130, 1-189, 1-290, 1-321, 1-369, 1-374
splitting	1-93, 1-163, 1-328
tape storage of	1-348
filters	1-57, 1-125, 1-177, 1-182, 1-251, 1-277, 1-299, 1-321, 1-364
FORTTRAN	1-9, 1-163, 1-293
group ID	1-41, 1-51, 1-58, 1-59, 1-161, 1-183, 1-213, 1-221, 1-255, 1-259
I/O,	1-35
options for	1-336
redirection of	1-75, 1-81, 1-306, 1-384
inter-process communication	1-100, 1-184, 1-185, 1-406
job control	1-68
libraries	1-19, 1-40, 1-192, 1-198, 1-207, 1-244
link editor	1-6, 1-160
links	1-59, 1-202, 1-212, 1-314
local registry	1-274
log-in	1-259, 1-274
logging in	1-67, 1-204, 1-206, 1-221, 1-306
macro processing	1-61, 1-107, 1-215, 1-239, 1-244, 1-251
mail	1-220, 1-223, 1-376, 1-382, 1-384, 1-406
mathematical functions	1-18, 1-31, 1-122, 1-159, 1-375
messages, sending	1-406

node ID	1-404
object files, size of	1-319
operators	1-76, 1-162, 1-194, 1-217
password	1-51, 1-204, 1-259, 1-274, 1-340
pattern matching	1-402
pattern scanning	1-13
patterns,	
matching of	1-59, 1-74, 1-177, 1-302
searching for	1-177
pipes	1-136, 1-139, 1-191, 1-251, 1-252, 1-266, 1-269, 1-299, 1-306, 1-355, 1-367
process ID	1-185
process IDs	1-287
processes,	
background	1-10, 1-82, 1-266, 1-400
child	1-400
parent	1-400
priority of	1-82, 1-262
remote	1-100, 1-220, 1-376, 1-379, 1-382, 1-384
status of	1-69, 1-287, 1-379
terminating	1-82, 1-191, 1-380, 1-400
suspending	1-400
programs, updating/regenerating	1-237
queries	1-363
RATFOR	1-163, 1-195, 1-215
Restricted Shell	1-140, 1-148, 1-306, 1-312, 1-317
SCCS files	1-127, 1-165, 1-244, 1-283, 1-297, 1-298, 1-299, 1-373, 1-387, 1-402,
SCCS,	
deltas	1-2, 1-44, 1-55
files	1-2, 1-55
numbers	1-127, 1-165, 1-166, 1-283, 1-297, 1-298, 1-373
security	1-274
servers	1-406
shell metacharacters	1-67, 1-71, 1-177, 1-376, 1-384
shell variables	1-83, 1-204, 1-206, 1-222, 1-224, 1-239, 1-251, 1-261, 1-295, 1-306
signals	1-82, 1-91, 1-191, 1-313
sizes	1-319
soft-copy	1-277
strings,	1-67, 1-77, 1-141, 1-157, 1-177, 1-194, 1-215, 1-217, 1-224, 1-239, 1-263, 1-364
deleting portions of	1-17

super-user	1-10, 1-50, 1-51, 1-59, 1-60, 1-64,1-89, 1-191, 1-340, 1-380
sync	1-343
terminfo database	1-363
UNIX-to-UNIX processing	1-220, 1-376, 1-379, 1-382, 1-384
user ID	1-49, 1-51, 1-58, 1-59, 1-104, 1-161, 1-183, 1-204, 1-213, 1-255, 1-287, 1-340, 1-404

intro – introduction to games	6-1
arithmetic – provide drill in number facts	6-2
back – the game of backgammon.....	6-3
bj – the game of black jack	6-4
craps – the game of craps.....	6-5
fish – play “Go Fish”	6-7
fortune – print a random comment	6-8
hangman – guess the word.....	6-9
mastermind – Mastermind guessing game	6-10
maze – generate a maze.....	6-11
moo – guessing game.....	6-12
number – convert Arabic numerals to English	6-13
psych – psychedelic design generator	6-14
quiz – test your knowledge.....	6-15
random – random number generator	6-16
trk – Star Trek game.....	6-17
ttt – tic-tac-toe	6-23
wump – the game of hunt-the-wumpus	6-24

NAME

intro – introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the directory */usr/games*.

NAME

arithmetic – provide drill in number facts

USAGE

`/usr/games/arithmetic [+-x/] [range]`

DESCRIPTION

Arithmetic presents simple arithmetic problems, and waits for you to type an answer. If the answer is correct, it replies “Right!”, and supplies a new problem. If the answer is wrong, it replies “What?”, until you respond correctly.

The first optional argument determines the kind of problem to be generated. A plus sign (+), minus sign (–), lowercase x, and a slash (/) produce addition, subtraction, multiplication, and division problems respectively. Specifying more than one of these characters on a command line generates a variety of problem types, all mixed in random order. Specifying any characters other than the four mentioned here also produces a random mix of problem types. If you specify no argument to **arithmetic**, subtraction problems appear by default.

The second optional argument is *range*, a decimal number. If used, all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to this number. The default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

Every twenty problems, it publishes statistics on correctness and the time required to answer. Specifically, the program tells you the number of correct and incorrect answers that you have given, as well as the total percentage of those correct. It also tells you how much time (in seconds) has elapsed, and the average number of seconds it took you to answer each problem. For example, the program may output something like this:

Rights 20; Wrongs 1; Score 95%
Total time 50 seconds; 2.5 seconds per problem

To quit the program, type an interrupt (↑Q).

NOTES

As a matter of educational philosophy, **arithmetic** does not supply correct answers, since the learner should be able to calculate them. Thus, it does not try to teach number facts, but instead serves as a drill program for those just past the first learning stage of arithmetic. Usually, the most relevant statistic it provides is time per problem, not percent correct.

NAME

back – the game of backgammon

USAGE

/usr/games/back

DESCRIPTION

Back is a program that provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent's level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You are also given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated in the rules of the game.

Points are numbered 1–24, with 1 being white's extreme inner table, 24 being brown's inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type **y** when **back** asks the question, "Instructions?", at the beginning of the game. When **back** first asks "Move?", type a question mark (?) to see a list of move options other than entering your numerical move.

When the game is finished, **back** asks you if you want the log. If you respond with a **y**, **back** attempts to append to or create a file called *back.log* in the current directory.

FILES

/usr/games/lib/backrules
rules file

*/tmp/b**
log temp file

back.log
log file

NOTES

Back complains loudly if you attempt to make too many moves in a turn, but becomes very silent if you make too few.

Doubling is not implemented.

NAME

bj – the game of black jack

USAGE

/usr/games/bj

DESCRIPTION

Bj is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is \$2 every hand.

A player “natural” (black jack) pays \$3. A dealer natural loses \$2. Both dealer and player naturals is a “push” (no money exchange).

If the dealer has an ace up, you can make an “insurance” bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where you win \$2 if the dealer has a natural, and lose \$1 if the dealer does not.

If dealt two cards of the same value, you can “double”, that is, play two hands, each with one of these cards. The bet also doubles (\$2 on each hand).

If a dealt hand totals 10 or 11, you may “double down”. This means that you may double the bet (\$2 to \$4) and receive exactly one more card on that hand.

Under normal play, you may “hit” (draw a card) as long as your total isn’t over twenty-one. If you “bust” (go over twenty-one), the dealer wins the bet.

When you “stand” (decide not to hit), the dealer hits until attaining a total of seventeen or more. If the dealer busts, you win the bet.

If both you and the dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions are asked at appropriate times. You must answer each question by a y and a carriage return for “yes”, or just a carriage return for “no”.

? (This means “do you want a hit?”)

Insurance?

Double down?

Every time the deck is shuffled, the dealer so states and the “action” (total bet) and “standing” (total won or lost) is printed. To exit, do an interrupt (↑Q) and the action and standing are printed.

NAME

craps – the game of craps

USAGE

/usr/games/craps

DESCRIPTION

Craps is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while you place bets. At any time, you may choose to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House; any other bet is a bet with the roller.

At the start of the game, you have a “bankroll” of \$2,000. The program begins prompting with:

bet?

The bet can be all or part of your bankroll. Any bet over the total bankroll is rejected and the program continues prompting until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (you win or lose, depending on whether the bet is placed with the roller or with the House; the odds are even). The first roll is the roll immediately following a bet:

1. On the first roll:

7 or 11 -- wins for the roller;

2, 3, or 12 -- wins for the House;

any other number is the point, so roll again (Rule 2 applies)

2. On subsequent rolls:

point -- roller wins;

7 -- House wins;

any other number -- roll again.

If you lose your entire bankroll, the House offers to lend you an additional \$2,000. The program prompts as follows:

marker?

A yes (or y) consummates the loan. Any other reply terminates the game.

If you owe the House money, the House reminds you, before you can place a bet, how many markers are outstanding.

If, at any time, you have outstanding markers and your bankroll exceeds \$2,000, the House asks:

Repay marker?

A reply of yes (or y) indicates your willingness to repay the loan. If only 1 marker is outstanding, the debt is immediately repaid. However, if more than 1 marker is outstanding, the House asks:

How many?

markers you want to repay. If you enter an invalid number or just a carriage return, the program prints an appropriate message and prompts with

How many?

until you provide a valid number.

If you accumulate 10 markers (a total of \$20,000 borrowed from the House), the program tells you and then exits.

Should your bankroll exceed \$50,000, the House automatically deducts from it the total amount of money needed to pay off all outstanding markers.

If you accumulate \$100,000 or more, you break the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

The program usually considers any reply other than a yes to be a no. Exceptions to this are when the program asks you if you want to place a bet (i.e., bet?) and when it asks how many markers you want to pay off (i.e., How many?).

To exit, send an interrupt (↑I). Before exiting, the program tells you whether you won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

NAME

fish – play “Go Fish”

USAGE

/usr/games/fish

DESCRIPTION

Fish plays the game of “Go Fish”, a childrens’ card game. The Object is to accumulate books of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card that is not in the second player’s hand: he replies “GO FISH!” The first player then draws a card from the “pool” of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying “p” as a first guess puts you into “pro” level (default level is very easy).

NAME

fortune – print a random comment

USAGE

/usr/games/fortune

DESCRIPTION

Fortune prints a fortune, anecdote, saying, or otherwise random comment. All lines are derived from the default fortune database in */usr/games/lib/fortunes.*

NAME

hangman – guess the word

USAGE

/usr/games/hangman [*arg*]

DESCRIPTION

Hangman chooses a word at least seven letters long from a dictionary. You must guess letters one at a time, until you guess the word.

The optional argument *arg* names an alternate dictionary.

FILES

/usr/lib/w2006 dictionary

NOTES

Hyphenated compounds are run together.

NAME

mastermind – Mastermind guessing game

USAGE

/usr/games/mastermind

DESCRIPTION

This program plays the game of "mastermind". The playing field is a number of slots, in which a number of colored pegs can be placed. The object of the game is to guess what color peg is in each slot. You and the program take turns trying to guess each other's configuration.

Before play begins, mastermind asks you whether you want instructions. Respond by typing a "y" for yes or an "n" for no. Following this, you have a chance to decide how many slots and how many colors you want to use. When you enter a guess, type the names of the colors, separated by spaces. When the program makes a guess, respond with two digits separated by spaces.

A guess consists of a possible sequence of colored pegs. The guesser's opponent answers with two numbers: the number of pegs in the guess that exactly match the corresponding pegs in the configuration, and the number of pegs in the guess that match in color but not in position. For example, suppose you are playing with five slots, and the following situation occurs:

my configuration:	red	red	yellow	blue	brown
your guess:	blue	red	green	red	red

The two numbers would then be 1 and 2. The 1 applies because both you and the program have a red peg in the second slot. In addition, your blue matches the program's blue, though the position is wrong, and one of your reds matches the program's red in the first slot. Only two of your reds match because the program only has two reds in its configuration.

Any time it is your turn to enter a guess, you can ask the program what happened by typing "review" instead of your guess. You get one point for each guess that the program has to make, and it gets one point for each guess that you have to make.

NAME

maze – generate a maze

USAGE

/usr/games/maze [*seed* [*n*]]

DESCRIPTION

Maze, with no arguments specified, prints a randomly selected maze, but shows no solution. Each maze uses a set of parameters called a *seed*. You may specify a particular seed, or have the program produce one at random. If you specify only a seed argument, **maze** prints the given maze with the solution to it. If you specify an *n* after the seed argument, the program prints the maze without a solution.

NOTES

Some mazes (especially small ones) have no solutions.

NAME

moo – guessing game

USAGE

/usr/games/moo

DESCRIPTION

Moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. Then, you guess four distinct digits being scored on each guess. A “cow” is a correct digit in an incorrect position. A “bull” is a correct digit in a correct position. The game continues until you guess the number (a score of four bulls).

NAME

number – convert Arabic numerals to English

USAGE

/usr/games/number

DESCRIPTION

Number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

psych – psychedelic design generator

USAGE

/usr/games/psych

DESCRIPTION

Psych is a menu-driven program that creates psychedelic patterns on your node. If you have a color node, you may select one of six different colors, black, or white for use in drawing a pattern. If your node is monochromatic, your choices are obviously limited to black or white lines. The patterns created by the program are always of random design.

When you first invoke **psych**, a menu appears in the left-hand side of the window running the program. To the right is a large blank area where the pattern you draw will appear. At the top of the menu are three boxes (EXIT, ERASE, and DRAW). Below that, there's a box for each of the eight LINE COLORS.

Using either a mouse, a touchpad, or the arrow keys on your node, place the cursor on a LINE COLOR. Select the color by pressing the left-most mouse key or hitting <F1> on your keyboard. You'll notice that the box surrounding the name of the color is becomes highlighted in reverse video. Now move your cursor to the box representing the DRAW function. Select DRAW in the same manner as the line color. A design will appear in the window. (Note that, though your design may contain any of the colors available to you, you can only select one color at a time.) You may then either repeat your action to add more lines to the design, ERASE the design, or EXIT the program.

If you need help, point your cursor at any of the boxes in the menu or on the window that hold the design. Press the middle key on your mouse, or hit <F2> on your keyboard. **Psych** responds by providing a brief description of the function of that item on the menu.

NAME

quiz – test your knowledge

USAGE

`/usr/games/quiz [-i file] [-t] [category1 category2]`

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*, or vice versa. If you don't specify a category, **quiz** gives instructions and lists the available categories.

If you don't know the answer to a question, simply hit a carriage return and **quiz** supplies a correct answer. At the end of input, upon interrupt, or when questions run out, **quiz** reports a score and terminates.

OPTIONS

-t Enter "tutorial" mode, where missed questions are repeated later, and material is gradually introduced as you learn.

-i file Substitute the named *file* for the default index file. The lines of these files have the syntax:

```
line = category new-line | category : line
category= alternate | category | alternate
alternate= empty | alternate primary
primary= character | [ category ] | option
option= { category }
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. A backslash (\) is used as with **sh(1)** to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, **quiz** refrains from asking it.

FILES

`/usr/games/lib/quiz/index`
`/usr/games/lib/quiz/*`

NOTES

The construct "a|ab" does not work in an information file. Use "a{b}".

NAME

random – random number generator

USAGE

/usr/games/random
[numbers] [characters]

DESCRIPTION

Random is a random number generator that accepts input of numbers or characters. There is about a 50% probability that the program will match and output the lines that you use as input. You can also perform a particular task (e.g., combining lines in a file, listing a directory, etc.) and then use **random** in a pipeline to produce a random sampling of the output. For example, this takes the contents of *mydir* and then lists a random percentage (about 50%) of the lines contained in it:

```
ls my_dir | random
```

Random is a likely candidate for use in other games that require the use of a random number generator.

NAME

trk - Star Trek game

USAGE

/usr/games/trk

DESCRIPTION

Trk is a game of space glory and war. Your mission is to travel through the galaxy, destroying Klingon ships before your time and resources expire. The game begins by asking you to choose the length of your playing session. Valid responses are: short, medium, and long. (Another possible response, restart, is currently unsupported.) You are then prompted for your skill level, to which you must respond novice, fair, good, expert, commadore, or impossible. In general, it is wise to choose *novice* and work your way up to the next skill level. After you have responded, the game asks you to enter a password. This password need not be the one that you enter at login. However, you should remember what you typed, because if you use the **destruct** command later in the game, the same password that you entered at the beginning of the game is required.

Following initial set-up, an arrangement similar to this appears on your screen:

20 Klingons; it takes 250 units to kill a Klingon.
6 starbases at 7,5 4,1 4,3 6,1 2,8 3,6

	0	1	2	3	4	5	6	7	8	9			
0	0	stardate	3300.00
1	1	condition	GREEN
2	E	2	position	5,8/2,4
3	3	warp factor	5.0
4	4	total energy	5000
5	*	5	torpedoes	10
6	6	shields	down, 100%
7	0	7	Klingons left	20
8	8	time left	26.00
9	9	life support	active
	0	1	2	3	4	5	6	7	8	9			

Your ship is the Enterprise (represented by the E). An enemy Klingon ship appears as a K on the screen. A B stands for a starbase, * identifies a star, 0 shows an inhabited star system, a blank represents a black hole, a dot is an empty space, and a Q stands for another ship called the Faerie Queene.

For a list of the available commands, type a question mark (?). The commands used by the game are summarized below. In most cases, if you forget what a valid response to a prompt should be, you can type a question mark and the game prints all possible responses. Usually, you can type a command and its argument all on the same line, instead of typing the command first and then waiting for another prompt for each argument.

OPTIONS

- a Scan on quadrant entry (autoscan).
- c Suppress the Command: prompt, and prompt with a simple colon (:).
- r Suppress random messages.

COMMAND SUMMARY

- ! Return to the shell.
- m Move the ship to another location. After typing **m**, you are prompted first for course and then for distance. Course is in degrees from 0-360; distance can be specified in either whole numbers or decimals. You may specify the move, along with the course and distance, all on the same line following the command prompt -- or you may choose to be prompted for each of these items. For example, suppose you are currently located in quadrant 4,3 and you want to move down to the quadrant 5,3. At the command prompt, simply type **m 180 1**. Note that this example is based on a traveling speed of *warp 5* (the default when you begin the game). Actual rate of movement may vary with your specified warp speed (see warp below).
- ram Ram a Klingon (or other object). This action is usually a last resort to destroy a Klingon ship, when you are under attack and have no weaponry left; its use is always risky, because of concurrent damages to your ship. After typing **ram**, you are prompted for sector and then sector quadrant of the Klingon ship. A damage report always appears following a ram. Take care in calculating these, as you may hit another object (e.g., starbase, etc.) and thus bring about total destruction of your ship.
- q Make a query. You can make individual inquiries about the *st*(ardate), *c*(ondition), *p*(osition), *w*(arp), *e*(nergy), *to*(rpedo), *sh*(ields), *k*(lingons), *t*(ime), *l*(ife), *cr*(ew), and *br*(ig). These items are the same ones listed when you enter a status command (see below).
- s Provide status information. When you type this command, you

get a listing that includes the current stardate, condition (red, yellow, green), position (quadrant, sector), warp factor, total energy remaining, torpedoes remaining, shield operability (percentage), number of Klingons left, amount of time left, amount of damage to life support systems (including reserves), number of crew members, and amount of space in the brig.

sh

Put shields up or down. After you type **sh**, the current status of the shields is reported (i.e., up or down). You are prompted for a response of *up* or *down*. The **sh** command and its argument may be specified on the same line.

s

Do a short-range scan. An arrangement similar to the one that appeared on your screen at the beginning of the game will be presented. This allows you to see the detail of the quadrant and sector in which your ship is currently located.

l

Do a long-range scan. This type of scan shows the Enterprise at the middle of the display, surrounded by adjacent quadrants. Consider, for example, the following:

	-1-	-2-	-3-	Long range scan for quadrant 7,2
6	3	3	9	
7	#3	E1	1	
8	5	101	7	

The right-most column always represents the number of stars and inhabited star systems in the quadrant, the middle column shows the Enterprise, and the left-most column identifies a star base if a pound sign (#) is shown, and the number of Klingon ships in the quadrant if a number is shown. Note that, in this example, 101 is shown in quadrant 8,2. The zero in the center separates the two ones, so that there is no question as to the number of stars and Klingon ships (1 each). The example can be interpreted as follows: three stars in quadrant 6,1 and 6,2; nine stars in quadrant 6,3; one star base and three stars in quadrant 7,1; Enterprise and one star in quadrant 7,2; one star in quadrant 7,3; five stars in quadrant 8,1; one Klingon ship and one star in quadrant 8,2; and seven stars in quadrant 8,3.

da

Provide damage report. This command lists the extent and nature of damages to the Enterprise. If no damages are apparent, it reports all devices functional.

p

Fire phasers. Generally, it takes 250 units of power to destroy a

Klingon ship. If your shields have been damaged and are not operating at full capacity, you will be told that power must be diverted from the shields in order to fire phasers. You may then choose manual or automatic mode for firing. If you choose automatic, phasers are automatically locked on a target. If you choose manual, they are not. In either case, you must specify the number of units to fire. A report on the consequences of your action immediately follows.

quit

Quit the game. The program asks you whether or not you would like to play another game. If you respond with a *yes* or *y*, a new set-up appears on your screen. If you respond with a *no* or *n*, the program aborts and returns you to the shell.

t

Fire a torpedo. After you type *t*, you are prompted for a torpedo course to which you must respond in degrees (same as when you move the ship). If you are unsure of the exact course, use the computer to calculate the trajectory for firing (see computer below). Once you have supplied the course, the program asks you whether or not you want a burst; such a burst of fire could help in circumstances where two Klingon ships are adjacent and you want to try to destroy both using one torpedo. You must, of course, specify a burst angle before the firing actually takes place. The results of the firing are reported immediately afterwards.

de

Plant a detonator for subsequent firing of a torpedo. After you type *de*, the program asks you to specify a course for the torpedo to follow, but does not fire the torpedo until you type the *t* command (see above). This means that the course is set, so when you do type *t* to fire the torpedo, you are not prompted for a course. Once a torpedo is planted to fire at a specified target, you cannot reset the target for that particular torpedo.

cl

Activate or deactivate cloaking device. This device makes the Enterprise invisible to Klingon ships. You may use it to move stealthily from one place to another, but you are warned that Federation regulations do not permit attack while cloaked. This doesn't mean that the program denies you the chance to fire upon ships while you are cloaked, but it does mean that you are penalized for doing so. After typing *cl*, you must indicate *up* or *down*.

c

Request information from the ship's computer. Available information includes: *r*(ecord) *f* the galaxy for all long-range scans; *m*(ove) to a specified quadrant and sector; *t*(rajectory) of Klingon

ships (including course and distance) in your immediate area; *c*(ourse) degrees and distance to reach a quadrant and sector that you specify; *s*(core) in number of Klingons killed, kill rate of Klingons/stardate, penalty for remaining Klingons or various violations, calls for help, and stars and star systems destroyed; *p*(haser effectiveness) at a given range; *w*(arp cost) for a given distance and warp factor; *i*(mpulse engine cost) for a given distance; *d*(istress list) of all star systems who have sent out calls for help during play; *b*(ase) location of nearest star base; *k*(alculator) for performing mathematical calculations; and *x* and *y* coordinate information.

- w** Set warp speed to increase or decrease amount of time to travel from one place to another. After you type **w**, the program prompts you for a warp factor. Although you are not prevented from typing any number, you are warned about the damaging consequences of specifying any warp factor greater than 6.
- r** Rest to allow time for necessary repairs. You should only use this command when the ship has been badly damaged. You may stop the ship where it is and make repairs, or you can try to get to a starbase if needed. After you type **r**, the program prompts you for the amount of time that you would like to rest. To help calculate the amount of time needed to make repairs, first get a damage report (see **da** command). Repairs are begun immediately; reports on those that get completed are then provided automatically.
- do** Dock the ship next to a starbase. This command resets and restores total energy (5000), number of torpedos (10), shield energy (100%), and number of current crew (387). In order to dock, you must have a starbase in your immediate quadrant, and you must be in the sector immediately adjacent to it. (If a starbase is not in sight, see the **help** command below.)
- i** Use impulse engines to move. After you type **i**, you are prompted for course and distance. Although the use of impulse engines sometimes consumes more total energy than warp power, it is helpful for zeroing in on a particular location, especially if you use the computer to calculate the exact course and distance to a particular area.
- u** Undock the ship from the starbase. Once you type this command, you are ready to move away from the starbase.
- help** Request help from a starbase. This command searches for an

available starbase, locates it, and tries to move your ship adjacent to it. Sometimes attempts for help turn out to be futile, and you receive a message attempt to rematerialize fails. The starbase makes a total of three attempts to move you; if the third try fails, the game ends and your score is reported. (Furthermore, if the ship's sub-space radio is out due to damages from attack, no attempt to locate a starbase will be made.) If the **help** succeeds, the Enterprise is docked next to the starbase.

v Perform a (very short-range) visual scan. The program prompts you for direction of the scan. Objects appearing at short distance in that direction are reported usually as question marks; unoccupied spaces appear as dots.

destruct Execute a self-destruct sequence. This command is useful for ending a game, when you want to get an automatic report of your total score (including penalties). It starts a count-down, and when it reaches the middle, prompts you for a password. This should be the same password that you entered at the beginning of the game. If it is not, the message self-destruct sequence aborted appears, and the game exits without reporting a score.

abandon Abandon ship. This action usually results in your being put in charge of an antiquated but still functional ship, the Faerie Queene. The capabilities of the Faerie Queene are more limited than those of the Enterprise, the most notable being reduced total energy (3000) and the lack of a shuttle craft (which means that you cannot **abandon** the Faerie Queene).

NAME

tft - tic-tac-toe

USAGE

/usr/games/tft

DESCRIPTION

Tft is the popular X and O game, but it is also a learning program that never makes the same mistake twice.

When you begin, the program prompts you with

Accumulated knowledge? (Yes or No)

You must respond with a yes or no. Then, the program tells you how many "words" of knowledge it currently has in its learning file. Then, tft states whether or not this is a "new game", prints a three-by-three grid of numbers 1-9, and prompts with "Your move?" as shown below:

new game

123

456

789

Your move?

One by one, as you specify numbers in the grid, the program places an "X" in the location where the number once was. After each turn that you have, the program takes a turn, placing an "O" in any available spot. This continues until either you or the program wins by creating a vertical, horizontal, or diagonal line of three of the same characters in a row on the grid. To exit the game, send an interrupt (↑I).

Although it learns, tft learns slowly. It must lose nearly 80 games to completely know the game.

FILES

/usr/games/tft.k learning file

NAME

wump – the game of hunt-the-wumpus

USAGE

/usr/games/wump

DESCRIPTION

Wump plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

At the beginning of the game, the program prompts you with

Instructions? (y-n)

If you want a more detailed description of the game, type a "y". Otherwise, type "n". After that, the program asks various questions which you answer, one per line. To exit, send an interrupt (↑I).

This program is based on one described in *People's Computer Company*, 2, 2 (November 1973).

NOTES

It will never replace Adventure.

Appendix A: Permuted Index

This permuted index covers reference material in the *DOMAIN/IX Command Reference Manual*, the *DOMAIN/IX Programmer's Reference Manual*, and parts of *System Administration for DOMAIN/IX*. In addition, there is a topical index located at the end of each section of these manuals.

	@: arithmetic on shell variables.....	csh(1)
	abort: generate a fault.....	abort(3)
	abs: integer absolute value.....	abs(3)
abs: integer	absolute value.....	abs(3)
fabs, floor, ceil:	absolute value, floor, ceiling functions.....	floor(3M)
accept:	accept a connection on a socket.....	accept(2)
	accept: accept a connection on a socket.....	accept(2)
	access: determine if a file can be accessed.....	access(2)
getgroups: get group	access list.....	getgroups(2)
initgroups: initialize group	access list.....	initgroups(3X)
setgroups: set group	access list.....	setgroups(2)
access: determine if a file can be	accessed.....	access(2)
pac: printer/plotter	accounting information.....	pac(8)
fix_cache - repair	acl cache hash chains.....	fix_cache(8)
flush_cache - clear the node's	acl_cache.....	flush_cache(8)
sin, cos, tan, asin,	acos, atan, atan2: trigonometric functions.....	sin(3M)
sact: print current SCCS file editing	activity.....	sact(1)
fortune: print a random	adage.....	fortune(6)
addroot:	add a root ID.....	addroot(8)
	addbib: create or extend bibliographic database.....	addbib(1)
inet_makeaddr, inet_lnaof, inet_netof: Internet	address manipulation routines. inet_ntoa,.....	inet(3n)
arp:	Address Resolution Protocol.....	arp(4P)
mailaddr: mail	addressing description.....	mailaddr(7)
	addroot: add a root ID.....	addroot(8)
	admin: create and administer SCCS files.....	admin(1)
admin: create and	administer SCCS files.....	admin(1)
intro: introduction to system	administration commands.....	intro(8)
update_slave: update auxiliary system	administrator's nodes.....	update_slave(8)
flock: place or remove an	advisory lock on an open file.....	flock(2)
yes: be repetitively	affirmative.....	yes(1)
basename: strip filename	affixes.....	basename(1)
crypt, encrypt: a one-way hashing encryption	algorithm.....	crypt(3)
	alias: shell macros.....	csh(1)
unalias: remove	aliases.....	csh(1)
	aliases: aliases file for sendmail.....	aliases(5)
which: locate a program file, including	aliases and paths.....	which(1)
newaliases: rebuild the database for the mail	aliases file.....	newaliases(1)
aliases:	aliases file for sendmail.....	aliases(5)

	valloc:	aligned memory allocator.	valloc(3)
	malloc, free, realloc, calloc,	alloca: memory allocator.	malloc(3)
malloc, free, realloc, calloc, alloca:	memory allocator.		malloc(3)
	valloc:	aligned memory allocator.	valloc(3)
	eyacc:	modified yacc allowing much improved error recovery.	eyacc(1)
	limit:	alter per-process resource limitations.	csh(1)
	renice:	alter priority of running processes.	renice(8)
	else:	alternative commands.	csh(1)
lex:	generator of lexical analysis programs.		lex(1)
	style:	analyze surface characteristics of a document.	style(1)
	tar:	tape (and general purpose) archiver.	tar(1)
	sigstack:	set and/or get signal stack context.	sigstack(2)
whereis:	locate binary and/or manual for program.		whereis(1)
	worms:	animate worms on a display terminal.	worms(6)
	rain:	animated raindrops display.	rain(6)
		a.out: cc output.	a.out(5)
	apply:	apply a command to a set of arguments.	apply(1)
		apply: apply a command to a set of arguments.	apply(1)
		apropos: locate commands by keyword lookup.	apropos(1)
	ar:	archive and library maintainer.	ar(1)
		ar: archive (library) file format.	ar(5)
	number:	convert Arabic numerals to English.	number(6)
	bc:	arbitrary-precision arithmetic language.	bc(1)
graphics	openpl, erase, label, line, circle,	arc, move, cont, point, linemod, space, closepl:	plot(3X)
	ar:	archive and library maintainer.	ar(1)
	tar:	tape archive file format.	tar(5)
	arcv:	convert archive files to new format.	arcv(8)
	ar:	archive (library) file format.	ar(5)
tar:	tape (and general purpose) archiver.		tar(1)
	ranlib:	convert archives to random libraries.	ranlib(1)
		arcv: convert archive files to new format.	arcv(8)
	glob:	filename expand argument list.	csh(1)
	shift:	manipulate argument list.	csh(1)
	varargs:	variable argument list.	varargs(3)
vsprintf:	print formatted output of a varargs argument list.	vprintf, vprintf,	vprintf(3S)
apply:	apply a command to a set of arguments.		apply(1)
	echo:	echo arguments.	csh(1)
	echo:	echo arguments.	echo(1)
	expr:	evaluate arguments as an expression.	expr(1)
	bc:	arbitrary-precision arithmetic language.	bc(1)
	@:	arithmetic on shell variables.	csh(1)
		arithmetic: provide drill in number facts.	arithmetic(6)
		arp: Address Resolution Protocol.	arp(4P)
	expr:	evaluate arguments as an expression.	expr(1)
gmtime, asctime, timezone:	convert date and time to ASCII.	ctime, localtime,	ctime(3)
	ascii:	map of ASCII character set.	ascii(7)
	od:	octal, decimal, hex, ASCII dump.	od(1)
		ascii: map of ASCII character set.	ascii(7)
	atof, atoi, atol:	convert ASCII to numbers.	atof(3)
	ctime, localtime, gmtime,	asctime, timezone: convert date and time to ASCII.	ctime(3)

sin, cos, tan,	asin, acos, atan, atan2: trigonometric functions.....	sin(3M)
help:	ask for help.....	help(1)
	assert: program verification.....	assert(3X)
setbuf, setbuffer, setlinebuf:	assign buffering to a stream.....	setbuf(3S)
setstate: better random number generator and	associated routines. random, srandom, initstate,	random(3)
nice, nohup: run a command	at a different priority.....	nice(1)
at: execute commands	at a later time.....	at(1)
	at: execute commands at a later time.....	at(1)
sin, cos, tan, asin, acos,	atan, atan2: trigonometric functions.....	sin(3M)
sin, cos, tan, asin, acos, atan,	atan2: trigonometric functions.....	sin(3M)
	atof, atoi, atol: convert ASCII to numbers.....	atof(3)
	atoi, atol: convert ASCII to numbers.....	atof(3)
	atol: convert ASCII to numbers.....	atof(3)
interrupt. sigpause:	atomically release blocked signals and wait for.....	sigpause(2)
update_slave: update	auxiliary system administrator's nodes.....	update_slave(8)
wait:	await completion of process.....	wait(1)
	awk: pattern scanning and processing language.....	awk(1)
backgammon: the game of	backgammon.....	backgammon(6)
	backgammon: the game of backgammon.....	backgammon(6)
bg: place job in	background.....	csh(1)
wait: wait for	background processes to complete.....	csh(1)
banner: print large	banner on printer.....	banner(6)
	banner: print large banner on printer.....	banner(6)
printcap: printer capability data	base.....	printcap(5)
vi: screen-oriented (visual) display editor	based on ex.....	vi(1)
	basename: strip filename affixes.....	basename(1)
	bc: arbitrary-precision arithmetic language.....	bc(1)
bcopy, bcmp, bzero, ffs:	bit and byte string operations.....	bstring(3)
operations.	bcopy, bcmp, bzero, ffs: bit and byte string.....	bstring(3)
cb: C program	beautifier.....	cb(1)
j0, j1, jn, y0, y1, yn:	Bessel functions.....	j0(3M)
routines. random, srandom, initstate, setstate:	better random number generator and associated	random(3)
	bg: place job in background.....	csh(1)
addbib: create or extend	bibliographic database.....	addbib(1)
roffbib: run off	bibliographic database.....	roffbib(1)
sortbib: sort	bibliographic database.....	sortbib(1)
index for a bibliography; find references in a	bibliography. indxbib, lookbib: build inverted.....	lookbib(1)
indxbib, lookbib: build inverted index for a	bibliography; find references in a bibliography.....	lookbib(1)
install: install	binaries.....	install(1)
whereis: locate	binary and/or manual for program.....	whereis(1)
uuencode,uudecode: encode/decode a	binary file for transmission via mail.....	uuencode(1C)
fread, fwrite: buffered	binary input/output.....	fread(3S)
bind:	bind a name to a socket.....	bind(2)
	bind: bind a name to a socket.....	bind(2)
	binmail: send or receive mail among users.....	binmail(1)
cp	/bin/start_csh: start a C shell.....	start_csh(1)
cp	/bin/start_sh: start a Bourne Shell.....	start_sh(1)
bcopy, bcmp, bzero, ffs:	bit and byte string operations.....	bstring(3)
sigblock:	block signals.....	sigblock(2)
sigpause: atomically release	blocked signals and wait for interrupt.....	sigpause(2)

sum: sum and count	blocks in a file.....	sum(1)
rc:	boot time shell script.	rc(8)
cp /bin/start_sh: start a	Bourne Shell.....	start_sh(1)
mille: play Mille	Bournes.....	mille(6)
switch: multi-way command	branch.	csh(1)
	break: exit while/foreach loop.	csh(1)
	breaksw: exit from switch.	csh(1)
fg:	bring job into foreground.....	csh(1)
	brk, sbrk: change data segment size.	brk(2)
fread, fwrite:	buffered binary input/output.	fread(3S)
stdio: standard	buffered input/output package.	intro(3S)
setbuf, setbuffer, setlinebuf: assign	buffering to a stream.....	setbuf(3S)
references in a bibliography. indxbib, lookbib:	build inverted index for a bibliography; find.....	lookbib(1)
ntohs: convert values between host and network	byte order. htonl, htons, ntohl,	byteorder(3n)
bcopy, bcmp, bzero, ffs: bit and	byte string operations.....	bstring(3)
swab: swap	bytes.....	swab(3)
bcopy, bcmp,	bzero, ffs: bit and byte string operations.	bstring(3)
cc:	C compiler.	cc(1)
cb:	C program beautifier.	cb(1)
indent: indent and format	C program source.....	indent(1)
lint: a	C program verifier.....	lint(1)
xstr: extract strings from	C programs to implement shared strings.	xstr(1)
cp /bin/start_csh: start a	C shell.....	start_csh(1)
mkstr: create an error message file by messaging	C source.....	mkstr(1)
hypot,	cabs: Euclidean distance.	hypot(3M)
fix_cache - repair acl	cache hash chains.....	fix_cache(8)
	cal: print calendar.	cal(1)
dc: desk	calculator.	dc(1)
cal: print	calendar.....	cal(1)
	calendar: reminder service.....	calendar(1)
malloc, free, realloc,	calloc, alloca: memory allocator.	malloc(3)
intro: introduction to system	calls and error numbers.	intro(2)
access: determine if a file	can be accessed.	access(2)
printcap: printer	capability data base.....	printcap(5)
termcap: terminal	capability database.	termcap(5)
cribbage: the	card game cribbage.....	cribbage(6)
	case: selector in switch.	csh(1)
	cat: catenate and print.....	cat(1)
ccat: compress and uncompress files, and then	cat them. compact, uncompact,	compact(1)
default:	catchall clause in switch.	csh(1)
cat:	catenate and print.....	cat(1)
	catman: format the files for this manual.....	catman(8)
	cb: C program beautifier.	cb(1)
	cc: C compiler.....	cc(1)
a.out:	cc output.	a.out(5)
them. compact, uncompact,	ccat: compress and uncompress files, and then cat.....	compact(1)
	cd: change directory.	csh(1)
	cd: change working directory.	cd(1)
	cdc: change the delta commentary of an SCCS delta.cdc(1)	
fabs, floor,	ceil: absolute value, floor, ceiling functions.	floor(3M)

fabs, floor, ceil: absolute value, floor,	ceiling functions.....	floor(3M)
fix_cache - repair acl cache hash	chains.....	fix_cache(8)
chdir:	change current working directory.....	chdir(2)
brk, sbrk:	change data segment size.....	brk(2)
default_acl:	change default file protection environment.....	default_acl(2)
cd:	change directory.....	csh(1)
chdir:	change directory.....	csh(1)
chgrp:	change group.....	chgrp(1)
passwd:	change log-in password.....	passwd(1)
chmod:	change mode.....	chmod(1)
chmod:	change mode of file.....	chmod(2)
umask:	change or display file creation mask.....	csh(1)
chown:	change owner or group of a file.....	chown(2)
cdc:	change the delta commentary of an SCCS delta.....	cdc(1)
rename:	change the name of a file.....	rename(2)
chown:	change the owner of files.....	chown(8)
ver:	change the version of Shell commands.....	ver(8)
delta: make a delta	(change) to an SCCS file.....	delta(1)
set:	change value of shell variable.....	csh(1)
cd:	change working directory.....	cd(1)
pipe: create an interprocess communication	channel.....	pipe(2)
ungetc: push	character back into input stream.....	ungetc(3S)
isspace, ispunct, isprint, iscntrl, isascii:	character classification macros. isdigit, isalnum, ...ctype(3)	
eqnchar: special	character definitions for eqn(1).....	eqnchar(7)
getc, getchar, fgetc, getw: get	character or word from stream.....	getc(3S)
putc, putchar, fputc, putw: put	character or word on a stream.....	putc(3S)
ascii: map of ASCII	character set.....	ascii(7)
style: analyze surface	characteristics of a document.....	style(1)
tr: translate	characters.....	tr(1)
	chdir: change current working directory.....	chdir(2)
	chdir: change directory.....	csh(1)
checkeq:	check files that use eqn(1) or neqn(1).....	checkeq(1)
checknr:	check nroff/troff files.....	checknr(1)
	checkeq: check files that use eqn(1) or neqn(1).....	checkeq(1)
	checknr: check nroff/troff files.....	checknr(1)
	chgrp: change group.....	chgrp(1)
	chmod: change mode.....	chmod(1)
	chmod: change mode of file.....	chmod(2)
	chown: change owner or group of a file.....	chown(2)
	chown: change the owner of files.....	chown(8)
closepl: graphics openpl, erase, label, line,	circle, arc, move, cont, point, linemod, space,.....	plot(3X)
ispunct, isprint, iscntrl, isascii: character	classification macros. isdigit, isalnum, isspace,.....	ctype(3)
default: catchall	clause in switch.....	csh(1)
uuclean: uucp spool directory	clean-up.....	uuclean(8C)
	clear: clear terminal screen.....	clear(1)
clear:	clear terminal screen.....	clear(1)
flush_cache:	clear the node's acl_cache.....	flush_cache(8)
ferro, feof,	clearerr, fileno: stream status inquiries.....	ferro(3S)
csh: a shell (command interpreter) with	C-like syntax.....	csh(1)
cron:	clock daemon.....	cron(8)

	close: delete a descriptor.	close(2)
fclose, fflush:	close or flush a stream.	fclose(3S)
opendir, readdir, telldir, seekdir, rewinddir,	closedir: directory operations.	directory(3)
circle, arc, move, cont, point, linemod, space,	closepl: graphics interface. erase, label, line,	plot(3X)
	cmp: compare two files.	cmp(1)
sccsfile: format of Source	Code Control System (SCCS) file	sccsfile(5)
	col: filter reverse line feeds.	col(1)
	colcrt: filter nroff output for CRT previewing.	colcrt(1)
	colrm: remove columns from a file.	colrm(1)
colrm: remove	columns from a file.	colrm(1)
	comb: combine SCCS deltas.	comb(1)
comb:	combine SCCS deltas.	comb(1)
files.	comm: select or reject lines common to two sorted	comm(1)
exec: overlay shell with specified	command.	csh(1)
time: time	command.	csh(1)
routines for returning a stream to a remote	command. rcmd, rresvport, ruserok:	rcmd(3X)
rexec: return stream to a remote	command.	rexec(3X)
system: issue a shell	command.	system(3)
test: condition	command.	test(1)
time: time a	command.	time(1)
nice, nohup: run a	command at a different priority.	nice(1)
switch: multi-way	command branch.	csh(1)
uux: UNIX-to-UNIX	command execution.	uux(1C)
rehash: recompute	command hash table.	csh(1)
unhash: discard	command hash table.	csh(1)
hashstat: print	command hashing statistics.	csh(1)
nohup: run	command immune to hangups.	csh(1)
csh: a shell	(command interpreter) with C-like syntax.	csh(1)
whatis: describe what a	command is.	whatis(1)
sh:	command language.	sh(1)
repeat: execute	command repeatedly.	csh(1)
onintr: process interrupts in	command scripts.	csh(1)
apply: apply a	command to a set of arguments.	apply(1)
goto:	command transfer.	csh(1)
else: alternative	commands.	csh(1)
intro: introduction to	commands.	intro(1)
intro: introduction to system administration	commands.	intro(8)
ver: change the version of Shell	commands.	ver(8)
at: execute	commands at a later time.	at(1)
apropos: locate	commands by keyword lookup.	apropos(1)
while: repeat	commands conditionally.	csh(1)
source: read	commands from file.	csh(1)
cdc: change the delta	commentary of an SCCS delta.	cdc(1)
comm: select or reject lines	common to two sorted files.	comm(1)
socket: create an endpoint for	communication.	socket(2)
pipe: create an interprocess	communication channel.	pipe(2)
users:	compact list of users who are on the system.	users(1)
files, and then cat them.	compact, uncompact, ccat: compress and uncompress	compact(1)
diff: differential file and directory	comparator.	diff(1)
cmp:	compare two files.	cmp(1)

	scsdiff: compare two versions of an SCCS file.....	scsdiff(1)
diff3: three-way differential file comparison.....		diff3(1)
intro: introduction to compatibility library functions.		intro(3C)
cc: C compiler.....		cc(1)
yacc: yet another compiler-compiler.		yacc(1)
wait: wait for background processes to complete.		csh(1)
wait: await completion of process.		wait(1)
compact, uncompact, ccat: compress and uncompress files, and then cat them.....		compact(1)
hangman: Computer version of the hangman game.....		hangman(6)
test: condition command.		test(1)
endif: terminate conditional.		csh(1)
if: conditional statement.		csh(1)
while: repeat commands conditionally.		csh(1)
inetd.conf: configuration file for inetd(8C).....		inetd.conf(5)
ifconfig: configure network interface parameters.		ifconfig(8C)
connect: initiate a connection on a socket.....		connect(2)
tip, cu: connect to a remote system.		cu(1C)
tip, cu: connect to a remote system.		tip(1C)
getpeername: get name of connected peer.....		getpeername(2)
socketpair: create a pair of connected sockets.....		socketpair(2)
shutdown: shut down part of a full-duplex socket connection.....		shutdown(2)
accept: accept a connection on a socket.....		accept(2)
connect: initiate a connection on a socket.....		connect(2)
listen: listen for connections on a socket.		listen(2)
deroff: remove nroff, troff, tbl, and eqn constructs.		deroff(1)
getrlimit: control maximum system resource consumption.....		getrlimit(2)
openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl: graphics.....		plot(3X)
ls: list contents of directory.		ls(1)
sigstack: set and/or get signal stack context.		sigstack(2)
continue: cycle in loop.		csh(1)
fcntl: file control.		fcntl(2)
ioctl: control device.		ioctl(2)
getrlimit: control maximum system resource consumption.....		getrlimit(2)
lpc: line printer control program.		lpc(8)
tcp: Internet Transmission Control Protocol.		tcp(4P)
scsfile: format of Source Code Control System (SCCS) file.....		scsfile(5)
term: conventional names for terminals.....		term(7)
ecvt, fcvt, gcvt: output conversion.....		ecvt(3)
printf, fprintf, sprintf: formatted output conversion.....		printf(3S)
scanf, fscanf, sscanf: formatted input conversion.....		scanf(3S)
units: conversion program.		units(1)
dd: convert and copy a file.		dd(1)
number: convert Arabic numerals to English.....		number(6)
arcv: convert archive files to new format.....		arcv(8)
ranlib: convert archives to random libraries.....		ranlib(1)
atof, atoi, atol: convert ASCII to numbers.....		atof(3)
ctime, localtime, gmtime, asctime, timezone: convert date and time to ASCII.		ctime(3)
cvtumap: convert name trees from SR8 to SR9 name mapping.....		cvtumap(8)
htable: convert NIC standard format host tables.....		htable(8)
htonl, htons, ntohl, ntohs: convert values between host and network byte order.....		byteorder(3n)

cp:	copy.	cp(1)
rcp: remote file	copy.	rcp(1C)
uucp, uuname, uulog: UNIX to UNIX	copy.	uucp(1C)
dd: convert and	copy a file.	dd(1)
functions. sin,	cos, tan, asin, acos, atan, atan2: trigonometric.	sin(3M)
sinh,	cosh, tanh: hyperbolic functions.	sinh(3M)
wc: word	count.	wc(1)
sum: sum and	count blocks in a file.	sum(1)
	cp /bin/start_csh: start a C shell.	start_csh(1)
	cp /bin/start_sh: start a Bourne Shell.	start_sh(1)
	cp: copy.	cp(1)
open: open a file for reading or writing, or	create a new file.	open(2)
	fork: create a new process.	fork(2)
socketpair:	create a pair of connected sockets.	socketpair(2)
ctags:	create a tags file.	ctags(1)
socket:	create an endpoint for communication.	socket(2)
mkstr:	create an error message file by massaging C source.	mkstr(1)
pipe:	create an interprocess communication channel.	pipe(2)
admin:	create and administer SCCS files.	admin(1)
mkdisk	create disk device descriptor files.	mkdisk(8)
soft_link, soft_unlink:	create or delete soft links.	soft_link(2)
addbib:	create or extend bibliographic database.	addbib(1)
crpasswd:	create password and group files.	crpasswd(8)
crpty:	create psuedo tty device entries.	crpty(8)
umask: change or display file	creation mask.	csh(1)
umask: set/get file	creation mask.	umask(2)
cribbage: the card game	cribbage.	cribbage(6)
	cribbage: the card game cribbage.	cribbage(6)
	cron: clock daemon.	cron(8)
	crpasswd: create password and group files.	crpasswd(8)
	crpty: create psuedo tty device entries.	crpty(8)
colcrt: filter nroff output for	CRT previewing.	colcrt(1)
more, page: file perusal filter for	CRT viewing.	more(1)
more, page: file perusal filter for	CRT viewing.	page(1)
algorithm.	crypt, encrypt: a one-way hashing encryption.	crypt(3)
syntax.	csh: a shell (command interpreter) with C-like.	csh(1)
locate a program file, including aliases and paths	which.	which(1)
	ctags: create a tags file.	ctags(1)
convert date and time to ASCII.	ctime, localtime, gmtime, asctime, timezone:	ctime(3)
tip,	cu: connect to a remote system.	cu(1C)
tip,	cu: connect to a remote system.	tip(1C)
gethostid, sethostid: get/set unique identifier of	current host.	gethostid(2)
gethostname, sethostname: get/set name of	current host.	gethostname(2)
hostid: set or print identifier of	current host system.	hostid(1)
hostname: set or print name of	current host system.	hostname(1)
jobs: print	current job list.	csh(1)
sact: print	current SCCS file editing activity.	sact(1)
sigsetmask: set	current signal mask.	sigsetmask(2)
whoami: print effective	current user ID.	whoami(1)
chdir: change	current working directory.	chdir(2)

getwd: get	current working directory pathname.	getwd(3)
motion.	curses: screen functions with optimized cursor	curses(3X)
curses: screen functions with optimized	cursor motion.....	curses(3X)
spline: interpolate smooth	curve.	spline(1G)
mapping.	cvtumap: convert name trees from SR8 to SR9 namecvtumap(8)	
continue:	cycle in loop.....	csh(1)
cron: clock	daemon.....	cron(8)
lpd: line printer	daemon.....	lpd(8)
routed: network routing	daemon.....	routed(8C)
writed:	daemon for write(1) program.	writed(8C)
ftpd:	DARPA Internet File Transfer Protocol server	ftpd(8C)
telnetd:	DARPA TELNET protocol server	telnetd(8C)
tftpd:	DARPA Trivial File Transfer Protocol server.....	tftpd(8C)
eval: re-evaluate shell	data.....	csh(1)
printcap: printer capability	data base.....	printcap(5)
brk, sbrk: change	data segment size.	brk(2)
null:	data sink.....	null(4)
types: primitive system	data types.....	types(5)
addbib: create or extend bibliographic	database.	addbib(1)
hosts: host name	database.	hosts(5)
networks: network name	database.	networks(5)
phones: remote host phone number	database.	phones(5)
protocols: protocol name	database.	protocols(5)
roffbib: run off bibliographic	database.	roffbib(1)
sortbib: sort bibliographic	database.	sortbib(1)
termcap: terminal capability	database.	termcap(5)
newaliases: rebuild the	database for the mail aliases file.	newaliases(1)
strfile: fortune(6)	database loader.	strfile(6)
services:	database of Internet services.....	services(5)
join: relational	database operator.....	join(1)
dbminit, fetch, store, delete, firstkey, nextkey:	database subroutines.....	dbm(3X)
udp: Internet User	Datagram Protocol.	udp(4P)
date: print the	date.....	date(1)
gettimeofday, settimeofday: get/set	date and time.....	gettimeofday(2)
localtime, gmtime, asctime, timezone: convert	date and time to ASCII. ctime,	ctime(3)
touch: update	date last modified of a file.....	touch(1)
	date: print the date.	date(1)
database subroutines.	dbminit, fetch, store, delete, firstkey, nextkey:.....	dbm(3X)
	dbx: debugger.....	dbx(1)
	dc: desk calculator.	dc(1)
	dd: convert and copy a file.	dd(1)
dbx:	debugger.	dbx(1)
od: octal,	decimal, hex, ASCII dump.	od(1)
	default: catchall clause in switch.	csh(1)
default_acl: change	default file protection environment	default_acl(2)
environment	default_acl: change default file protection.....	default_acl(2)
eqnchar: special character	definitions for eqn(1).....	eqnchar(7)
close:	delete a descriptor.	close(2)
dbminit, fetch, store,	delete, firstkey, nextkey: database subroutines.	dbm(3X)
soft_link, soft_unlink: create or	delete soft links.	soft_link(2)

	tail:	deliver the last part of a file.....	tail(1)
cdc:	change the delta commentary of an SCCS	delta.....	cdc(1)
	delta: make a	delta (change) to an SCCS file.....	delta(1)
	cdc: change the	delta commentary of an SCCS delta.....	cdc(1)
	rmel: remove a	delta from an SCCS file.....	rmel(1)
		delta: make a delta (change) to an SCCS file.....	delta(1)
comb:	combine SCCS	deltas.....	comb(1)
	mesg: permit or	deny messages.....	mesg(1)
	constructs.	deroff: remove nroff, troff, tbl, and eqn.....	deroff(1)
	whatis:	describe what a command is.....	whatis(1)
mailaddr:	mail addressing	description.....	mailaddr(7)
	remote: remote host	description file.....	remote(5)
	close: delete a	descriptor.....	close(2)
dup, dup2:	duplicate a	descriptor.....	dup(2)
mkdisk -	create disk device	descriptor files.....	mkdisk(8)
	getdtablesize: get	descriptor table size.....	getdtablesize(2)
	dc:	desk calculator.....	dc(1)
	file:	determine file type.....	file(1)
	access:	determine if a file can be accessed.....	access(2)
fold:	fold long lines for finite width output	device.....	fold(1)
	ioctl: control	device.....	ioctl(2)
	mkdisk -	device descriptor files.....	mkdisk(8)
	crpty: create psuedo tty	device entries.....	crpty(8)
	mtio: tape	device files.....	mtio(4)
		df: disk free.....	df(1)
	ratfor: rational FORTRAN	dialect.....	ratfor(1)
explain:	print wordy sentences; thesaurus for	diction.....	diction(1)
	for diction.	diction, explain: print wordy sentences; thesaurus.....	diction(1)
		diff: differential file and directory comparator.....	diff(1)
		diff3: three-way differential file comparison.....	diff3(1)
nice, nohup:	run a command at a	different priority.....	nice(1)
	diff:	differential file and directory comparator.....	diff(1)
	diff3: three-way	differential file comparison.....	diff3(1)
		dir: format of directories.....	dir(5)
	directories.....	directories.....	dir(5)
rm, rmdir:	remove (unlink)	directories or files.....	rm(1)
	cd: change working	directory.....	cd(1)
chdir:	change current working	directory.....	chdir(2)
	cd: change	directory.....	csh(1)
	chdir: change	directory.....	csh(1)
ls:	list contents of	directory.....	ls(1)
	mkdir: make a	directory.....	mkdir(1)
	scandir: scan a	directory.....	scandir(3)
uuclean:	uucp spool	directory clean-up.....	uuclean(8C)
diff:	differential file and	directory comparator.....	diff(1)
	unlink: remove	directory entry.....	unlink(2)
	mkdir: make a	directory file.....	mkdir(2)
	rmdir: remove a	directory file.....	rmdir(2)
	pwd: working	directory name.....	pwd(1)
readdir, telldir, seekdir, rewinddir, closedir:		directory operations. opendir.....	directory(3)

getwd: get current working	directory pathname.....	getwd(3)
popd: pop shell	directory stack.....	csh(1)
pushd: push shell	directory stack.....	csh(1)
unhash:	discard command hash table.....	csh(1)
unset:	discard shell variables.....	csh(1)
synchronize a file's in-core state with that on	disk. fsync:.....	fsync(2)
mkdisk - create	disk device descriptor files.....	mkdisk(8)
df:	disk free.....	df(1)
du: summarize	disk usage.....	du(1)
mount, umount: mount and	dismount file system.....	mount(8)
rain: animated raindrops	display.....	rain(6)
vi: screen-oriented (visual)	display editor based on ex.....	vi(1)
umask: change or	display file creation mask.....	csh(1)
man:	display reference manual information.....	man(1)
man:	display reference manual information.....	man.1.11(12)
worms: animate worms on a	display terminal.....	worms(6)
systype:	display version stamp.....	systype(8)
hypot, cabs: Euclidean	distance.....	hypot(3M)
style: analyze surface characteristics of a	document.....	style(1)
refer: find and insert literature references in	documents.....	refer(1)
shutdown: shut	down part of a full-duplex socket connection.....	shutdown(2)
graph:	draw a graph.....	graph(1G)
arithmetic: provide	drill in number facts.....	arithmetic(6)
pty: pseudo terminal	driver.....	pty(4)
od: octal, decimal, hex, ASCII	du: summarize disk usage.....	du(1)
	dump.....	od(1)
	dup, dup2: duplicate a descriptor.....	dup(2)
	dup2: duplicate a descriptor.....	dup(2)
	dup, dup2: duplicate a descriptor.....	dup(2)
	echo: echo arguments.....	csh(1)
	echo: echo arguments.....	echo(1)
	echo: echo arguments.....	csh(1)
	echo: echo arguments.....	echo(1)
	ecvt, fcvt, gcvt: output conversion.....	ecvt(3)
	ed: text editor.....	ed(1)
	end, etext, edata: last location in program.....	end(3)
	ex, edit: text editor.....	ex(1)
sact: print current SCCS file	editing activity.....	sact(1)
ed: text	editor.....	ed(1)
ex, edit: text	editor.....	ex(1)
ld: link	editor.....	ld(1)
sed: stream	editor.....	sed(1)
vi: screen-oriented (visual) display	editor based on ex.....	vi(1)
whoami: print	effective current user ID.....	whoami(1)
setregid: set real and	effective group ID.....	setregid(2)
setreuid: set real and	effective user ID.....	setreuid(2)
vfork: spawn a new process in a more	efficient way.....	vfork(2)
grep,	egrep, fgrep: search a file for a pattern.....	grep(1)
insque, remque: insert or remove an	element in a queue.....	insque(3)
soelim:	eliminate .so's from nroff input.....	soelim(1)

uencode: format of an mail. uencode,uudecode:	else: alternative commands.....csh(1) encoded uencode file.....uencode(5) encode/decode a binary file for transmission viauencode(1C)
crypt, crypt, encrypt: a one-way hashing	encrypt: a one-way hashing encryption algorithm. .crypt(3) encryption algorithm.crypt(3)
logout:	end, etext, edata: last location in program.....end(3) end session.....csh(1) end: terminate loop.csh(1)
getgrent, getgrgid, getgrnam, setgrent, gethostbyaddr, gethostbyname, sethostent,	endgrent: get group file entry.....getgrent(3) endhostent: get network host entry. gethostent,gethostent(3n) endif: terminate conditional.....csh(1)
getnetent, getnetbyaddr, getnetbyname, setnetent, socket: create an	endnetent: get network entry.getnetent(3n) endpoint for communication.....socket(2)
getprotobynumber, getprotobyname, setprotoent, getpwent, getpwuid, getpwnam, setpwent, getservbyport, getservbyname, setservent,	endprotoent: get protocol entry. getprotoent,getprotoent(3n) endpwent: get password file entry.....getpwent(3) endservent: get service entry. getservent,getservent(3n) endsw: terminate switch.....csh(1)
number: convert Arabic numerals to crpty: create psuedo tty device manx: macros for formatting	English.number(6) entries.....crpty(8) entries in this manual.....manx(7)
getgmam, setgrent, endgrent: get group file sethostent, endhostent: get network host getnetbyname, setnetent, endnetent: get network setprotoent, endprotoent: get protocol getpwnam, setpwent, endpwent: get password file getservbyname, setservent, endservent: get service unlink: remove directory	entry. getgrent, getgrgid,getgrent(3) entry. gethostent, gethostbyaddr, gethostbyname, ..gethostent(3n) entry. getnetent, getnetbyaddr,getnetent(3n) entry. getprotobynumber, getprotobyname,getprotoent(3n) entry. getpwent, getpwuid,getpwent(3) entry. getservent, getservbyport,getservent(3n) entry.unlink(2) environ: environment variables.environ(7) environ: execute a file.....execl(3) environment.csh(1) environment.....default_acl(2) environment.printenv(1) environment variable.....getenv(3) environment variables.csh(1) environment variables.environ(7) eqn constructs.deroff(1) eqn: format mathematical text for troff.eqn(1) eqn(1).....eqnchar(7) eqn(1) or neqn(1).checkeq(1) eqnchar: special character definitions for eqn(1)....eqnchar(7) erase, label, line, circle, arc, move, cont, point,.....plot(3X) error message file by massaging C source.....mkstr(1) error messages.perror(3) error numbers.intro(2) error recovery.eyacc(1) errors.spell(1) etext, edata: last location in program.....end(3) Euclidean distance.....hypot(3M) eval: re-evaluate shell data.csh(1) expr: evaluate arguments as an expression.expr(1)

history: print history	event list.csh(1)
screen-oriented (visual) display editor based on	ex. vi:.....vi(1)
lpq: spool queue	ex, edit: text editor.....ex(1)
	examination program.lpq(1)
	exec: overlay shell with specified command.csh(1)
environ: execute a file.	execl, execv, execl, execlp, execvp, exect,.....execl(3)
file. execl, execv,	execl, execlp, execvp, exect, environ: execute a file.....execl(3)
execl, execv, execl,	execlp, execvp, exect, environ: execute a file.execl(3)
execl, execv, execl, execlp, execvp,	exect, environ: execute a file.....execl(3)
execv, execl, execlp, execvp, exect, environ:	execute a file. execl,.....execl(3)
execve:	execute a file.execve(2)
repeat:	execute command repeatedly.....csh(1)
at:	execute commands at a later time.....at(1)
uux: UNIX-to-UNIX command	execution.....uux(1C)
sleep: suspend	execution for an interval.sleep(1)
sleep: suspend	execution for interval.sleep(3)
rexecd: remote	execution serverrexecd(8C)
execute a file. execl,	execv, execl, execlp, execvp, exect, environ:execl(3)
	execve: execute a file.....execve(2)
execl, execv, execl, execlp,	execvp, exect, environ: execute a file.execl(3)
breaksw:	exit from switch.csh(1)
	exit: leave shell.csh(1)
	_exit: terminate a process.exit(2)
pending output.	exit: terminate a process after flushing any.....exit(3)
break:	exit while/foreach loop.csh(1)
power, square root.	exp, log, log10, pow, sqrt: exponential, logarithm, exp(3M)
glob: filename	expand argument list.csh(1)
expand, unexpand:	expand tabs to spaces and vice versa.....expand(1)
versa.	expand, unexpand: expand tabs to spaces and vice expand(1)
diction. diction,	explain: print wordy sentences; thesaurus fordiction(1)
frexp, ldexp, modf: split into mantissa and	exponent.....frexp(3)
exp, log, log10, pow, sqrt:	exponential, logarithm, power, square root.....exp(3M)
	expr: evaluate arguments as an expression.....expr(1)
expr: evaluate arguments as an	expression.expr(1)
re_comp, re_exec: regular	expression handler.....regex(3)
addbib: create or	extend bibliographic database.....addbib(1)
strings. xstr:	extract strings from C programs to implement sharedxstr(1)
recovery.	eyacc: modified yacc allowing much improved error eyacc(1)
functions.	fabs, floor, ceil: absolute value, floor, ceilingfloor(3M)
networking: introduction to networking	facilities.intro(4N)
signal: simplified software signal	facilities.signal(3C)
sigvec: software signal	facilities.sigvec(2)
arithmetic: provide drill in number	facts.....arithmetic(6)
true,	false: provide truth values.true(1)
	false, true: provide truth values.false(1)
inet: Internet protocol	family.....inet(4F)
abort: generate a	fault.abort(3)
	fclose, fflush: close or flush a stream.....fclose(3S)
	fcntl: file control.....fcntl(2)
ecvt,	fcvt, gcvt: output conversion.ecvt(3)

fopen, freopen,	fdopen: open a stream.....	fopen(3S)
col: filter reverse line	feeds.....	col(1)
ferror,	feof, clearerr, fileno: stream status inquiries.....	ferror(3S)
inquiries.	ferror, feof, clearerr, fileno: stream status.....	ferror(3S)
subroutines. dbminit,	fetch, store, delete, firstkey, nextkey: database	dbm(3X)
head: give first	few lines.	head(1)
fclose,	fflush: close or flush a stream.....	fclose(3S)
bcopy, bcmp, bzero,	ffs: bit and byte string operations.	bstring(3)
	fg: bring job into foreground.....	csh(1)
getc, getchar,	fgetc, getw: get character or word from stream.	getc(3S)
gets,	fgets: get a string from a stream.	gets(3S)
grep, egrep,	fgrep: search a file for a pattern.....	grep(1)
chmod: change mode of	file.	chmod(2)
chown: change owner or group of a	file.	chown(2)
colrm: remove columns from a	file.	colrm(1)
source: read commands from	file.	csh(1)
ctags: create a tags	file.	ctags(1)
dd: convert and copy a	file.	dd(1)
delta: make a delta (change) to an SCCS	file.	delta(1)
execle, execlp, execvp, exect, environ: execute a	file. execl, execv,	execl(3)
execve: execute a	file.	execve(2)
flock: place or remove an advisory lock on an open	file.	flock(2)
fpr: print FORTRAN	file.	fpr(1)
get: get a version of an SCCS	file.	get(1)
group: group	file.	group(5)
link: make a hard link to a	file.	link(2)
mkdir: make a directory	file.	mkdir(2)
mknod: make a special	file.	mknod(2)
rebuild the database for the mail aliases	file. newaliases:	newaliases(1)
open a file for reading or writing, or create a new	file. open:	open(2)
passwd: password	file.	passwd(5)
pr: print	file.	pr(1)
prs: print an SCCS	file.	prs(1)
remote: remote host description	file.	remote(5)
rename: change the name of a	file.	rename(2)
rev: reverse lines of a	file.	rev(1)
rmel: remove a delta from an SCCS	file.	rmel(1)
rmdir: remove a directory	file.	rmdir(2)
sccsdiff: compare two versions of an SCCS	file.	sccsdiff(1)
format of Source Code Control System (SCCS)	file sccsfile:	sccsfile(5)
size: size of an object	file.	size(1)
strings: find the printable strings in an object	file.	strings(1)
symbol and line number information from an object	file. strip: strip	strip(1)
sum: sum and count blocks in a	file.	sum(1)
symlink: make symbolic link to a	file.	symlink(2)
tail: deliver the last part of a	file.	tail(1)
touch: update date last modified of a	file.	touch(1)
unget: undo a previous get of an SCCS	file.	unget(1)
uniq: report repeated lines in a	file.	uniq(1)
uuencode: format of an encoded uuencode	file.	uuencode(5)

val: validate SCCS	file.....val(1)
write, writev: write on a	file.....write(2)
diff: differential	file and directory comparator.....diff(1)
mkstr: create an error message	file by massaging C source.....mkstr(1)
access: determine if a	file can be accessed.....access(2)
diff3: three-way differential	file comparison.....diff3(1)
fcntl:	file control.....fcntl(2)
rcp: remote	file copy.....rcp(1C)
umask: change or display	file creation mask.....csh(1)
umask: set/get	file creation mask.....umask(2)
sact: print current SCCS	file: determine file type.....file(1)
getgrgid, getgrnam, setgrent, endgrent: get group	file editing activity.....sact(1)
getpwnam, setpwent, endpwent: get password	file entry. getgrent.....getgrent(3)
grep, egrep, fgrep: search a	file entry. getpwent, getpwuid.....getpwent(3)
inetd.conf: configuration	file for a pattern.....grep(1)
open: open a	file for inetd(8C).....inetd.conf(5)
aliases: aliases	file for reading or writing, or create a new file.....open(2)
uuencode,uudecode: encode/decode a binary	file for sendmail.....aliases(5)
ar: archive (library)	file for transmission via mail.....uuencode(1C)
tar: tape archive	file format.....ar(5)
intro: introduction to	file format.....tar(5)
which: locate a program	file formats.....intro(5)
fsplit: split a multi-routine FORTRAN	file, including aliases and paths.....which(1)
split: split a	file into individual files.....fsplit(1)
more, page:	file into pieces.....split(1)
more, page:	file perusal filter for CRT viewing.....more(1)
default_acl: change default	file perusal filter for CRT viewing.....page(1)
stat, lstat, fstat: get	file protection environment.....default_acl(2)
mount, umount: mount or remove	file status.....stat(2)
mount, umount: mount and dismount	file system.....mount(2)
hier:	file system.....mount(8)
mtab: mounted	file system hierarchy.....hier(7)
utimes: set	file system table.....mtab(5)
uuseed: send a	file times.....utimes(2)
truncate: truncate a	file to a remote host.....uuseed(1C)
ftp:	file to a specified length.....truncate(2)
ftpd: DARPA Internet	file transfer program.....ftp(1C)
tftpd: DARPA Trivial	File Transfer Protocol server.....ftpd(8C)
file: determine	File Transfer Protocol server.....tftpd(8C)
mktemp: make a unique	file type.....file(1)
basename: strip	filename.....mktemp(3)
glob:	filename affixes.....basename(1)
ferror, feof, clearerr,	filename expand argument list.....csh(1)
admin: create and administer SCCS	fileno: stream status inquiries.....ferror(3S)
checknr: check nroff/troff	files.....admin(1)
chown: change the owner of	files.....checknr(1)
cmp: compare two	files.....chown(8)
comm: select or reject lines common to two sorted	files.....cmp(1)
crpasswd: create password and group	files.....comm(1)
	files.....crpasswd(8)

find: find	files.....	find(1)
split a multi-routine FORTRAN file into individual	files. fsplit:.....	fsplit(1)
special files: introduction to special	files.....	intro(4)
mkdisk - create disk device descriptor	files.....	mkdisk(8)
mtio: tape device	files.....	mtio(4)
mv: move or rename	files.....	mv(1)
rm, rmdir: remove (unlink) directories or	files.....	rm(1)
sort: sort or merge	files.....	sort(1)
what: identify SCCS	files.....	what(1)
compact, uncompact, ccat: compress and uncompress	files, and then cat them.....	compact(1)
catman: format the	files for this manual	catman(8)
fsync: synchronize a	file's in-core state with that on disk.....	fsync(2)
special	files: introduction to special files.	intro(4)
lpr: print	files off-line.	lpr(1)
checkeq: check	files that use eqn(1) or neqn(1).	checkeq(1)
arcv: convert archive	files to new format	arcv(8)
fstab: static information about	filesystems.	fstab(5)
more, page: file perusal	filter for CRT viewing.	more(1)
more, page: file perusal	filter for CRT viewing.	page(1)
colcrt:	filter nroff output for CRT previewing.	colcrt(1)
col:	filter reverse line feeds.....	col(1)
plot: graphics	filters.	plot(1G)
refer:	find and insert literature references in documents..	refer(1)
find:	find files.	find(1)
	find: find files.	find(1)
look:	find lines in a sorted list.	look(1)
ttyname, isatty:	find name of a terminal.	ttyname(3)
lorder:	find ordering relation for an object library.	lorder(1)
lookbib: build inverted index for a bibliography;	find references in a bibliography. indxib,.....	lookbib(1)
spell, spellin, spellout:	find spelling errors.	spell(1)
strings:	find the printable strings in an object file.....	strings(1)
fold: fold long lines for	finite width output device.	fold(1)
head: give	first few lines.....	head(1)
dbminit, fetch, store, delete,	firstkey, nextkey: database subroutines.	dbm(3X)
fish: play "Go	Fish"	fish(6)
	fish: play "Go Fish".	fish(6)
tee: pipe	fitting.....	tee(1)
	fix_cache - repair acl cache hash chains.....	fix_cache(8)
file.	flock: place or remove an advisory lock on an open	flock(2)
functions. fabs,	floor, ceil: absolute value, floor, ceiling	floor(3M)
fabs, floor, ceil: absolute value,	floor, ceiling functions.	floor(3M)
fclose, fflush: close or	flush a stream.	fclose(3S)
	flush_cache - clear the node's acl_cache.	flush_cache(8)
exit: terminate a process after	flushing any pending output.	exit(3)
	fmit: simple text formatter.	fmit(1)
device.	fold: fold long lines for finite width output.....	fold(1)
fold:	fold long lines for finite width output device.	fold(1)
	fopen, freopen, fdopen: open a stream.....	fopen(3S)
	foreach: loop over list of names.....	csh(1)
fg: bring job into	foreground.	csh(1)

	fork: create a new process.....	fork(2)
ar: archive (library) file	format.....	ar(5)
arcv: convert archive files to new	format.....	arcv(8)
tar: tape archive file	format.....	tar(5)
indent: indent and	format C program source.....	indent(1)
htable: convert NIC standard	format host tables.....	htable(8)
gettable: get NIC	format host tables from a host.....	gettable(8C)
eqn:	format mathematical text for troff.....	eqn(1)
uencode:	format of an encoded uencode file.....	uencode(5)
dir:	format of directories.....	dir(5)
scsfile:	format of Source Code Control System (SCCS) files.....	scsfile(5)
tbl:	format tables for nroff or troff.....	tbl(1)
catman:	format the files for this manual.....	catman(8)
intro: introduction to file	formats.....	intro(5)
scanf, fscanf, sscanf:	formatted input conversion.....	scanf(3S)
printf, fprintf, sprintf:	formatted output conversion.....	printf(3S)
vprintf, vfprintf, vsprintf: print	formatted output of a varargs argument list.....	vprintf(3S)
fmt: simple text	formatter.....	fmt(1)
nroff: text	formatting.....	nroff(1)
troff: text	formatting and typesetting.....	troff(1)
manx: macros for	formatting entries in this manual.....	manx(7)
ms: text	formatting macros.....	ms(7)
man: macros for	formatting manual pages.....	man(7)
me: macros for	formatting papers.....	me(7)
ratfor: rational	FORTTRAN dialect.....	ratfor(1)
fpr: print	FORTTRAN file.....	fpr(1)
fsplit: split a multi-routine	FORTTRAN file into individual files.....	fsplit(1)
	fortune: print a random adage.....	fortune(8)
strfile:	fortune(6) database loader.....	strfile(6)
	fpr: print FORTRAN file.....	fpr(1)
printf,	fprintf, sprintf: formatted output conversion.....	printf(3S)
putc, putchar,	fputc, putw: put character or word on a stream.....	putc(3S)
puts,	fputs: put a string on a stream.....	puts(3S)
	fread, fwrite: buffered binary input/output.....	fread(3S)
df: disk	free.....	df(1)
malloc,	free, realloc, calloc, alloca: memory allocator.....	malloc(3)
fopen,	freopen, fdopen: open a stream.....	fopen(3S)
exponent.	frexp, ldexp, modf: split into mantissa and.....	frexp(3)
from: who is my mail	from?.....	from(1)
scanf,	fscanf, sscanf: formatted input conversion.....	scanf(3S)
	fseek, ftell, rewind: reposition a stream.....	fseek(3S)
individual files.	fsplit: split a multi-routine FORTRAN file into.....	fsplit(1)
	fstab: static information about filesystems.....	fstab(5)
stat, lstat,	fstat: get file status.....	stat(2)
on disk.	fsync: synchronize a file's in-core state with that.....	fsync(2)
fseek,	ftell, rewind: reposition a stream.....	fseek(3S)
	ftp: file transfer program.....	ftp(1C)
	ftpd: DARPA Internet File Transfer Protocol server.....	ftpd(8C)
shutdown: shut down part of a	full-duplex socket connection.....	shutdown(2)
gamma: log gamma	function.....	gamma(3M)

fabs, floor, ceil: absolute value, floor, ceiling	functions. floor(3M)
intro: introduction to library	functions. intro(3)
intro: introduction to compatibility library	functions. intro(3C)
intro: introduction to mathematical library	functions. intro(3M)
intro: introduction to network library	functions. intro(3n)
intro: introduction to miscellaneous library	functions. intro(3X)
j0, j1, jn, y0, y1, yn: Bessel	functions. j0(3M)
cos, tan, asin, acos, atan, atan2: trigonometric	functions. sin, sin(3M)
sinh, cosh, tanh: hyperbolic	functions. sinh(3M)
curses: screen	functions with optimized cursor motion. curses(3X)
fread,	fwrite: buffered binary input/output. fread(3S)
hangman: Computer version of the hangman	game. hangman(6)
trek: trekkie	game. trek(6)
worm: Play the growing worm	game. worm(6)
cribbage: the card	game cribbage. cribbage(6)
backgammon: the	game of backgammon. backgammon(6)
intro: introduction to	games. intro(6)
gamma: log	gamma function. gamma(3M)
ecvt, fcvt,	gamma: log gamma function. gamma(3M)
abort:	gcvt: output conversion. ecvt(3)
srandom, initstate, setstate: better random number	generate a fault. abort(3)
lex:	generator and associated routines. random, random(3)
from stream.	generator of lexical analysis programs. lex(1)
stream. getc,	getc, getchar, fgetc, getw: get character or word getc(3S)
getgid,	getchar, fgetc, getw: get character or word from getc(3S)
getuid,	getdtablesize: get descriptor table size. getdtablesize(2)
get group file entry.	getgid: get group identity. getgid(2)
file entry. getgrent,	getenv: get the value of an environment variable. getenv(3)
getgrent, getgrgid,	geteuid: get user identity. getuid(2)
endhostent: get network host entry. gethostent,	getgid, getegid: get group identity. getgid(2)
host entry. gethostent, gethostbyaddr,	getgrent, getgrgid, getgmam, setgrent, endgrent: getgrent(3)
sethostent, endhostent: get network host entry.	getgrgid, getgmam, setgrent, endgrent: get group getgrent(3)
current host.	getgmam, setgrent, endgrent: get group file entry. getgrent(3)
host.	getgroups: get group access list. getgroups(2)
timer.	gethostbyaddr, gethostbyname, sethostent, gethostent(3n)
get network entry. getnetent,	gethostbyname, sethostent, endhostent: get network gethostent(3n)
entry. getnetent, getnetbyaddr,	gethostent, gethostbyaddr, gethostbyname, gethostent(3n)
endnetent: get network entry.	gethostid, sethostid: get/set unique identifier of gethostid(2)
	gethostname, sethostname: get/set name of current gethostname(2)
	getitimer, setitimer: get/set value of interval. getitimer(2)
	getlogin: get log-in name. getlogin(3)
	getnetbyaddr, getnetbyname, setnetent, endnetent: getnetent(3n)
	getnetbyname, setnetent, endnetent: get network getnetent(3n)
	getnetent, getnetbyaddr, getnetbyname, setnetent, getnetent(3n)
	getpagesize: get system page size. getpagesize(2)
	getpass: read a password. getpass(3)
	getpeername: get name of connected peer. getpeername(2)
	getpgrp: get process group. getpgrp(2)
	getpid, getppid: get process identification. getpid(2)
	getppid: get process identification. getpid(2)

scheduling priority.	getpriority, setpriority: get/set program	getpriority(2)
protocol entry. getprotoent, getprotobyname,	getprotobyname, setprotoent, endprotoent: get	getprotoent(3n)
endprotoent: get protocol entry. getprotoent,	getprotobyname, getprotobyname, setprotoent,	getprotoent(3n)
setprotoent, endprotoent: get protocol entry.	getprotoent, getprotobyname, getprotobyname, ...	getprotoent(3n)
get password file entry.	getpwent, getpwuid, getpwnam, setpwent, endpwent:	getpwent(3)
entry. getpwent, getpwuid,	getpwnam, setpwent, endpwent: get password file.	getpwent(3)
password file entry. getpwent,	getpwuid, getpwnam, setpwent, endpwent: get	getpwent(3)
consumption.	getrlimit: control maximum system resource	getrlimit(2)
utilization.	getrusage: get information about resource	getrusage(2)
	gets, fgets: get a string from a stream.	gets(3S)
entry. getservent, getservbyport,	getservbyname, setservent, endservent: get service.	getservent(3n)
endservent: get service entry. getservent,	getservbyport, getservbyname, setservent,	getservent(3n)
setservent, endservent: get service entry.	getservent, getservbyport, getservbyname,	getservent(3n)
gettimeofday, settimeofday:	get/set date and time.	gettimeofday(2)
gethostname, sethostname:	get/set name of current host.	gethostname(2)
getsockopt, setsockopt:	get/set options on sockets.	getsockopt(2)
getpriority, setpriority:	get/set program scheduling priority.	getpriority(2)
gethostid, sethostid:	get/set unique identifier of current host.	gethostid(2)
getitimer, setitimer:	get/set value of interval timer.	getitimer(2)
	getsockname: get socket name.	getsockname(2)
	getsockopt, setsockopt: get/set options on sockets.	getsockopt(2)
	gettable: get NIC format host tables from a host....	gettable(8C)
	gettimeofday, settimeofday: get/set date and time.	gettimeofday(2)
	getuid, geteuid: get user identity.	getuid(2)
getc, getchar, fgetc,	getw: get character or word from stream.....	getc(3S)
	getwd: get current working directory pathname.....	getwd(3)
head:	give first few lines.....	head(1)
	glob: filename expand argument list.	csh(1)
ASCII. ctime, localtime,	gmtime, asctime, timezone: convert date and time	toctime(3)
fish: play	"Go Fish"	fish(6)
setjmp, longjmp: non-local	goto.	setjmp(3)
	goto: command transfer.	csh(1)
graph: draw a	graph.	graph(1G)
	graph: draw a graph.	graph(1G)
plot:	graphics filters.	plot(1G)
arc, move, cont, point, linemod, space, closepl:	graphics interface. erase, label, line, circle,	plot(3X)
plot:	graphics interface.	plot(5)
	grep, egrep, fgrep: search a file for a pattern.	grep(1)
chgrp: change	group.	chgrp(1)
getpgrp: get process	group.	getpgrp(2)
killpg: send signal to a process	group.	killpg(2)
setpgrp: set process	group.	setpgrp(2)
getgroups: get	group access list.	getgroups(2)
initgroups: initialize	group access list.	initgroups(3X)
setgroups: set	group access list.	setgroups(2)
group:	group file.	group(5)
getgrgid, getgrnam, setgrent, endgrent: get	group file entry. getgrent,	getgrent(3)
crpasswd: create password and	group files.	crpasswd(8)
	group: group file.	group(5)
setuid setgid setegid setrgid: set user and	group ID setuid seteuid	net(3n)

setregid: set real and effective	group ID.	setregid(2)
setruid, setgid, setegid, setrgid: set user and	group ID. setuid, seteuid,	setuid(3)
getgid, getegid: get	group identity.	getgid(2)
groups: show	group memberships.	groups(1)
chown: change owner or	group of a file.	chown(2)
make: maintain program	groups.	make(1)
	groups: show group memberships.	groups(1)
worm: Play the	growing worm game.	worm(6)
stop:	halt a job or process.	csh(1)
reboot: reboot system or	halt processor.	reboot(2)
	halt: stop the processor.	halt(8)
rmail:	handle remote mail received via uucp.	rmail(1)
re_comp, re_exec: regular expression	handler.	regex(3)
hangman: Computer version of the	hangman: Computer version of the hangman game.	hangman(6)
nohup: run command immune to	hangman game.	hangman(6)
link: make a	hangups.	csh(1)
fix_cache - repair acl cache	hard link to a file.	link(2)
rehash: recompute command	hash chains.	fix_cache(8)
unhash: discard command	hash table.	csh(1)
crypt, encrypt: a one-way	hash table.	csh(1)
hashstat: print command	hashing encryption algorithm.	crypt(3)
	hashing statistics.	csh(1)
leave: remind you when you	hashstat: print command hashing statistics.	csh(1)
help: ask for	have to leave.	leave(1)
	help.	help(1)
	help: ask for help.	help(1)
od: octal, decimal,	hex, ASCII dump.	od(1)
	hier: file system hierarchy.	hier(7)
hier: file system	hierarchy.	hier(7)
history: print	history event list.	csh(1)
	history: print history event list.	csh(1)
sethostid: get/set unique identifier of current	host. gethostid,	gethostid(2)
gethostname, sethostname: get/set name of current	host.	gethostname(2)
gettable: get NIC format host tables from a	host.	gettable(8C)
uusend: send a file to a remote	host.	uusend(1C)
htonl, htons, ntohl, ntohs: convert values between	host and network byte order.	byteorder(3n)
remote: remote	host description file.	remote(5)
gethostbyname, sethostent, endhostent: get network	host entry. gethostent, gethostbyaddr,	gethostent(3n)
hosts:	host name database.	hosts(5)
phones: remote	host phone number database.	phones(5)
ruptime: show	host status of local machines.	ruptime(1C)
hostid: set or print identifier of current	host system.	hostid(1)
hostname: set or print name of current	host system.	hostname(1)
htable: convert NIC standard format	host tables.	htable(8)
gettable: get NIC format	host tables from a host.	gettable(8C)
system.	hostid: set or print identifier of current host.	hostid(1)
	hostname: set or print name of current host system.	hostname(1)
	hosts: host name database.	hosts(5)
uptime: show	how long a node has been up.	uptime(1)
	htable: convert NIC standard format host tables.	htable(8)

host and network byte order.	htonl, htons, ntohl, ntohs: convert values between..byteorder(3n)
and network byte order. htonl,	htons, ntohl, ntohs: convert values between host....byteorder(3n)
sinh, cosh, tanh:	hyperbolic functions.....sinh(3M)
	hypot, cabs: Euclidean distance.....hypot(3M)
addroot: add a root	ID.....addroot(8)
setruid setgid setegid setrgid: set user and group	ID setuid seteuid.....net(3n)
setregid: set real and effective group	ID.....setregid(2)
setreuid: set real and effective user	ID.....setreuid(2)
setgid, setegid, setrgid: set user and group	ID. setuid, seteuid, setruid,.....setuid(3)
whoami: print effective current user	ID.....whoami(1)
su: substitute user	ID temporarily.....su(1)
getpid, getppid: get process	identification.....getpid(2)
gethostid, sethostid: get/set unique	identifier of current host.....gethostid(2)
hostid: set or print	identifier of current host system.....hostid(1)
what:	identify SCCS files.....what(1)
getgid, getegid: get group	identity.....getgid(2)
getuid, geteuid: get user	identity.....getuid(2)
access: determine	if a file can be accessed.....access(2)
	if: conditional statement.....csh(1)
notify: request	ifconfig: configure network interface parameters....ifconfig(8C)
nohup: run command	immediate notification.....csh(1)
xstr: extract strings from C programs to	immune to hangups.....csh(1)
eyacc: modified yacc allowing much	implement shared strings.....xstr(1)
which: locate a program file,	improved error recovery.....eyacc(1)
fsync: synchronize a file's	including aliases and paths.....which(1)
indent:	in-core state with that on disk.....fsync(2)
	indent and format C program source.....indent(1)
tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal	indent: indent and format C program source.....indent(1)
ptx: permuted	independent operation routines. tgetent,.....termcap(3X)
bibliography. indxbib, lookbib: build inverted	index.....ptx(1)
strncat, strcmp, strncmp, strcpy, strncpy, strlen,	index for a bibliography; find references in a.....lookbib(1)
fsplit: split a multi-routine FORTRAN file into	index, rindex: string operations. strcat,.....string(3)
bibliography; find references in a bibliography.	individual files.....fsplit(1)
	indxbib, lookbib: build inverted index for a.....lookbib(1)
inet_lnaof, inet_netof: Internet address	inet: Internet protocol family.....inet(4F)
	inet_addr, inet_network, inet_ntoa, inet_makeaddr,inet(3n)
inetd.conf: configuration file for	inetd: Internet superdaemon.....inetd(8C)
	inetd(8C).....inetd.conf(5)
inet_addr, inet_network, inet_ntoa, inet_makeaddr,	inetd.conf: configuration file for inetd(8C).....inetd.conf(5)
address inet_addr, inet_network, inet_ntoa,	inet_lnaof, inet_netof: Internet address.....inet(3n)
inet_network, inet_ntoa, inet_makeaddr, inet_lnaof,	inet_makeaddr, inet_lnaof, inet_netof: Internet.....inet(3n)
inet_netof: Internet address inet_addr,	inet_netof: Internet address manipulation routines.inet(3n)
Internet address inet_addr, inet_network,	inet_network, inet_ntoa, inet_makeaddr, inet_lnaof,inet(3n)
man: display reference manual	inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof:....inet(3n)
man: display reference manual	information.....man(1)
pac: printer/plotter accounting	information.....man.1.11(12)
fstab: static	information.....pac(8)
getrusage: get	information about filesystems.....fstab(5)
strip: strip symbol and line number	information about resource utilization.....getrusage(2)
	information from an object file.....strip(1)

intro: miscellaneous useful	information pages.....	intro(7)
tset: terminal-dependent	initgroups: initialize group access list.....	initgroups(3X)
initgroups:	initialization.....	tset(1)
connect:	initgroups: initialize group access list.....	initgroups(3X)
popen, pclose:	initiate a connection on a socket.....	connect(2)
and associated routines. random, srandom,	initiate I/O to and from a process.....	popen(3)
read, readv: read	initstate, setstate: better random number generator.....	random(3)
soelim: eliminate .so's from nroff	input.....	read(2)
scanf, fscanf, sscanf: formatted	input.....	soelim(1)
ungetc: push character back into	input conversion.....	scanf(3S)
fread, fwrite: buffered binary	input stream.....	ungetc(3S)
stdio: standard buffered	input/output.....	fread(3S)
ferror, feof, clearerr, fileno: stream status	input/output package.....	intro(3S)
refer: find and	inquiries.....	ferror(3S)
insque, remque:	insert literature references in documents.....	refer(1)
queue.	insert or remove an element in a queue.....	insque(3)
install:	insque, remque: insert or remove an element in a.....	insque(3)
	install binaries.....	install(1)
	install: install binaries.....	install(1)
cont, point, linemod, space, closepl: graphics	interface. erase, label, line, circle, arc, move,.....	plot(3X)
plot: graphics	interface.....	plot(5)
tty: general terminal	interface.....	tty(4)
ifconfig: configure network	interface parameters.....	ifconfig(8C)
telnet: user	interface to the TELNET protocol.....	telnet(1C)
sendmail: send mail over the	internet.....	sendmail(8)
inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof:	Internet address manipulation routines.....	inet(3n)
ftpd: DARPA	Internet File Transfer Protocol server.....	ftpd(8C)
inet:	Internet protocol family.....	inet(4F)
services: database of	Internet services.....	services(5)
inetd:	Internet superdaemon.....	inetd(8C)
tcp:	Internet Transmission Control Protocol.....	tcp(4P)
udp:	Internet User Datagram Protocol.....	udp(4P)
spline:	interpolate smooth curve.....	spline(1G)
csh: a shell (command	interpreter) with C-like syntax.....	csh(1)
pipe: create an	interprocess communication channel.....	pipe(2)
atomically release blocked signals and wait for	interrupt. sigpause:.....	sigpause(2)
onintr: process	interrupts in command scripts.....	csh(1)
sleep: suspend execution for an	interval.....	sleep(1)
sleep: suspend execution for	interval.....	sleep(3)
intro:	introduction to commands.....	intro(1)
intro:	introduction to compatibility library functions.....	intro(3C)
intro:	introduction to file formats.....	intro(5)
intro:	introduction to games.....	intro(6)
intro:	introduction to library functions.....	intro(3)
intro:	introduction to mathematical library functions.....	intro(3M)
intro:	introduction to miscellaneous library functions.....	intro(3X)
intro:	introduction to network library functions.....	intro(3n)
networking:	introduction to networking facilities.....	intro(4N)
special files:	introduction to special files.....	intro(4)
intro:	introduction to system administration commands.....	intro(8)

intro:	introduction to system calls and error numbers.....	intro(2)
in a bibliography. indxib, lookbib:	build inverted index for a bibliography; find references..	lookbib(1)
select:	synchronous I/O multiplexing	select(2)
popen, pclose:	initiate I/O to and from a process.....	popen(3)
	ioctl: control device.	ioctl(2)
whatis:	describe what a command is.....	whatis(1)
isascii: isalpha, isupper, islower, isdigit,	isalnum, isspace, ispunct, isprint, iscntrl,.....	ctype(3)
isspace, ispunct, isprint, iscntrl, isascii:	isalpha, isupper, islower, isdigit, isalnum,	ctype(3)
isalnum, isspace, ispunct, isprint, iscntrl,	isascii: character classification macros. isdigit,	ctype(3)
ttyname,	isatty: find name of a terminal.	ttyname(3)
isdigit, isalnum, isspace, ispunct, isprint,	iscntrl, isascii: character classification macros.	ctype(3)
iscntrl, isascii: isalpha, isupper, islower,	isdigit, isalnum, isspace, ispunct, isprint,	ctype(3)
isprint, iscntrl, isascii: isalpha, isupper,	islower, isdigit, isalnum, isspace, ispunct,	ctype(3)
islower, isdigit, isalnum, isspace, ispunct,	isprint, iscntrl, isascii: character classification.....	ctype(3)
isupper, islower, isdigit, isalnum, isspace,	ispunct, isprint, iscntrl, isascii: character	ctype(3)
isalpha, isupper, islower, isdigit, isalnum,	isspace, ispunct, isprint, iscntrl, isascii:	ctype(3)
system:	issue a shell command.	system(3)
ispunct, isprint, iscntrl, isascii: isalpha,	isupper, islower, isdigit, isalnum, isspace,	ctype(3)
j0,	j0, j1, jn, y0, y1, yn: Bessel functions.	j0(3M)
j0, j1,	j1, jn, y0, y1, yn: Bessel functions.	j0(3M)
bg: place	job in background.	csh(1)
fg: bring	job into foreground.	csh(1)
jobs: print current	job list.	csh(1)
stop: halt a	job or process.	csh(1)
kill: kill	jobs and processes.	csh(1)
lprm: remove	jobs from the line printer spooling queue.....	lprm(1)
	jobs: print current job list	csh(1)
	join: relational database operator.	join(1)
msgs: system messages and	junk mail program.....	msgs(1)
apropos: locate commands by	keyword lookup.....	apropos(1)
kill:	kill jobs and processes.	csh(1)
	kill: kill jobs and processes.	csh(1)
	kill: send signal to a process.	kill(2)
	kill: terminate a specified process.	kill(1)
	killpg: send signal to a process group.....	killpg(2)
linemod, space, closepl: graphics	label, line, circle, arc, move, cont, point,	plot(3X)
awk: pattern scanning and processing	language.....	awk(1)
bc: arbitrary-precision arithmetic	language.....	bc(1)
sh: command	language.....	sh(1)
	ld: link editor.....	ld(1)
frexp,	ldexp, modf: split into mantissa and exponent.	frexp(3)
leave: remind you when you have to	leave.	leave(1)
	leave: remind you when you have to leave.	leave(1)
exit:	leave shell.....	csh(1)
truncate: truncate a file to a specified	length.	truncate(2)
	lex: generator of lexical analysis programs.	lex(1)
lex: generator of	lexical analysis programs.....	lex(1)
ranlib: convert archives to random	libraries.	ranlib(1)
lorder: find ordering relation for an object	library.....	lorder(1)

ar: archive	(library) file format.	ar(5)
intro: introduction to	library functions.	intro(3)
intro: introduction to compatibility	library functions.	intro(3C)
intro: introduction to mathematical	library functions.	intro(3M)
intro: introduction to network	library functions.	intro(3n)
intro: introduction to miscellaneous	library functions.	intro(3X)
ar: archive and	library maintainer.	ar(1)
limit: alter per-process resource	limitations.	limit(1)
unlimit: remove resource	limitations.	unlimit(1)
space, closepl: graphics openpl, erase, label,	line, circle, arc, move, cont, point, linemod,	plot(3X)
col: filter reverse	line feeds.	col(1)
strip: strip symbol and	line number information from an object file.	strip(1)
print: pr to the	line printer.	print(1)
lpc:	line printer control program.	lpc(8)
lpd:	line printer daemon.	lpd(8)
lprm: remove jobs from the	line printer spooling queue.	lprm(1)
erase, label, line, circle, arc, move, cont, point,	linemod, space, closepl: graphics interface.	plot(3X)
head: give first few	lines.	head(1)
comm: select or reject	lines common to two sorted files.	comm(1)
fold: fold long	lines for finite width output device.	fold(1)
uniq: report repeated	lines in a file.	uniq(1)
look: find	lines in a sorted list.	look(1)
rev: reverse	lines of a file.	rev(1)
readlink: read value of a symbolic	link.	readlink(2)
ld:	link editor.	ld(1)
link: make a hard	link: make a hard link to a file.	link(2)
symlink: make symbolic	link to a file.	link(2)
ln: make	link to a file.	symlink(2)
soft_link, soft_unlink: create or delete soft	links.	ln(1)
glob: filename expand argument	links.	soft_link(2)
history: print history event	lint: a C program verifier.	lint(1)
jobs: print current job	list.	list(1)
shift: manipulate argument	list.	list(1)
getgroups: get group access	list.	list(1)
initgroups: initialize group access	list.	list(1)
look: find lines in a sorted	list.	list(1)
nm: print name	list.	list(1)
setgroups: set group access	list.	list(1)
varargs: variable argument	list.	list(1)
print formatted output of a varargs argument	list.	list(1)
ls:	list contents of directory.	ls(1)
foreach: loop over	list of names.	list(1)
users: compact	list of users who are on the system.	users(1)
listen:	listen for connections on a socket.	listen(2)
refer: find and insert	listen: listen for connections on a socket.	listen(2)
	literature references in documents.	refer(1)
	ln: make links.	ln(1)

strfile: fortune(6) database	loader.strfile(6)
and time to ASCII. ctime,	localtime, gmtime, asctime, timezone: convert date(3)
which:	locate a program file, including aliases and paths...which(1)
whereis:	locate binary and/or manual for program.whereis(1)
apropos:	locate commands by keyword lookup.apropos(1)
end, etext, edata: last	location in program.end(3)
flock: place or remove an advisory	lock on an open file.flock(2)
gamma:	log gamma function.gamma(3M)
power, square root. exp,	log, log10, pow, sqrt: exponential, logarithm,exp(3M)
syslog:	log systems messages.syslog(8)
square root. exp, log,	log10, pow, sqrt: exponential, logarithm, power,exp(3M)
exp, log, log10, pow, sqrt: exponential,	logarithm, power, square root.exp(3M)
rwho: who's	logged in on local machines.rwho(1C)
rlogin: remote	log-in.rlogin(1C)
	login: login new user.csh(1)
getlogin: get	log-in name.getlogin(3)
login:	login new user.csh(1)
passwd: change	log-in password.passwd(1)
rlogind: remote	log-in server.rlogind(8C)
	login: sign on.login(1)
	logout: end session.csh(1)
setjmp,	longjmp: non-local goto.setjmp(3)
	look: find lines in a sorted list.look(1)
find references in a bibliography. indxib,	lookbib: build inverted index for a bibliography; ...lookbib(1)
apropos: locate commands by keyword	lookup.apropos(1)
break: exit while/foreach	loop.csh(1)
continue: cycle in	loop.csh(1)
end: terminate	loop.csh(1)
foreach:	loop over list of names.csh(1)
library.	lorder: find ordering relation for an objectlorder(1)
	lpc: line printer control program.lpc(8)
	lpd: line printer daemon.lpd(8)
	lpq: spool queue examination program.lpq(1)
	lpr: print files off-line.lpr(1)
queue.	lprm: remove jobs from the line printer spooling...lprm(1)
	ls: list contents of directory.ls(1)
	lseek: move read/write pointer.lseek(2)
stat,	lstat, fstat: get file status.stat(2)
	m4: macro processor.m4(1)
runtime: show host status of local	machines.runtime(1C)
rwho: who's logged in on local	machines.rwho(1C)
m4:	macro processor.m4(1)
alias: shell	macros.csh(1)
isprint, iscntrl, isascii: character classification	macros. isdigit, isalnum, isspace, ispunct,ctype(3)
ms: text formatting	macros.ms(7)
manx:	macros for formatting entries in this manual.manx(7)
man:	macros for formatting manual pages.man(7)
me:	macros for formatting papers.me(7)
mt:	magnetic tape manipulating program.mt(1)
mail: send and receive	mail.mail(1)

encode/decode a binary file for transmission via	mail. uuencode,uudecode:	uuencode(1C)
mailaddr:	mail addressing description.	mailaddr(7)
newaliases: rebuild the database for the	mail aliases file.....	newaliases(1)
binmail: send or receive	mail among users.	binmail(1)
from: who is my	mail from?	from(1)
prmail: print out	mail in the post office.....	prmail(1)
sendmail: send	mail over the internet.....	sendmail(8)
msgs: system messages and junk	mail program.	msgs(1)
mail: handle remote	mail received via uucp.....	rmail(1)
	mail: send and receive mail.....	mail(1)
	mailaddr: mail addressing description.....	mailaddr(7)
make:	maintain program groups.	make(1)
ar: archive and library	maintainer.	ar(1)
delta:	make a delta (change) to an SCCS file.....	delta(1)
mkdir:	make a directory.....	mkdir(1)
mkdir:	make a directory file.....	mkdir(2)
link:	make a hard link to a file.	link(2)
mknod:	make a special file.	mknod(2)
mktemp:	make a unique filename.....	mktemp(3)
ln:	make links.	ln(1)
	make: maintain program groups.....	make(1)
symlink:	make symbolic link to a file.....	symlink(2)
script:	make typescript of a terminal session.....	script(1)
allocator.	malloc, free, realloc, calloc, alloca: memory	malloc(3)
	man: display reference manual information.	man(1)
	man: display reference manual information.	man.1.11(12)
	man: macros for formatting manual pages..	man(7)
shift:	manipulate argument list.....	csh(1)
route: manually	manipulate the routing tables.	route(8C)
mt: magnetic tape	manipulating program.	mt(1)
inet_lnaof, inet_netof: Internet address	manipulation routines. inet_ntoa, inet_makeaddr, ..	inet(3n)
frexp, ldexp, modf: split into	mantissa and exponent.	frexp(3)
catman: format the files for this	manual	catman(8)
manx: macros for formatting entries in this	manual.	manx(7)
whereis: locate binary and/or	manual for program.	whereis(1)
man: display reference	manual information.	man(1)
man: display reference	manual information.	man.1.11(12)
man: macros for formatting	manual pages.....	man(7)
route:	manually manipulate the routing tables.	route(8C)
	manx: macros for formatting entries in this manual.	manx(7)
cvtumap: convert name trees from SR8 to SR9 name	mapping.	cvtumap(8)
umask: change or display file creation	mask.	csh(1)
sigsetmask: set current signal	mask.	sigsetmask(2)
umask: set/get file creation	mask.	umask(2)
mkstr: create an error message file by	massaging C source.....	mkstr(1)
intro: introduction to	mathematical library functions.....	intro(3M)
eqn: format	mathematical text for troff.....	eqn(1)
getrlimit: control	maximum system resource consumption.	getrlimit(2)
	me: macros for formatting papers.	me(7)
groups: show group	memberships.....	groups(1)

malloc, free, realloc, calloc, alloca:	memory allocator.	malloc(3)
valloc: aligned	memory allocator.	valloc(3)
sort: sort or	merge files.	sort(1)
mkstr: create an error	msg: permit or deny messages.	msg(1)
recv, recvfrom, recvmsg: receive a	message file by massaging C source.	mkstr(1)
send, sendto, sendmsg: send a	message from a socket.	recv(2)
msg: permit or deny	message from a socket.	send(2)
perror, sys_errlist, sys_nerr: system error	messages.	msg(1)
psignal, sys_siglist: system signal	messages.	perror(3)
syslog: log systems	messages.	psignal(3)
msgs: system	messages.	syslog(8)
mille: play	messages and junk mail program.	msgs(1)
intro: introduction to	Mille Bournes.	mille(6)
intro:	mille: play Mille Bournes.	mille(6)
mkdir: make a directory.	miscellaneous library functions.	intro(3X)
mkdir: make a directory file.	miscellaneous useful information pages.	intro(7)
mkdisk - create disk device descriptor files.	mkdir: make a directory.	mkdir(1)
mknod: make a special file.	mkdir: make a directory file.	mkdir(2)
mkstr: create an error message file by massaging Cmkstr(1)	mkdisk - create disk device descriptor files.	mkdisk(8)
mktemp: make a unique filename.	mknod: make a special file.	mknod(2)
mode.	mkstr: create an error message file by massaging Cmkstr(1)	mktemp(3)
chmod: change	mktemp: make a unique filename.	mktemp(3)
chmod: change	mode.	chmod(1)
frexp, ldexp,	mode of file.	chmod(2)
touch: update date last	modf: split into mantissa and exponent.	frexp(3)
recovery. eyacc:	modified of a file.	touch(1)
vfork: spawn a new process in a	modified yacc allowing much improved error.	eyacc(1)
	more efficient way.	vfork(2)
	more, page: file perusal filter for CRT viewing.	more(1)
	more, page: file perusal filter for CRT viewing.	page(1)
curses: screen functions with optimized cursor	motion.	curses(3X)
mount, umount:	mount and dismount file system.	mount(8)
mount, umount:	mount or remove file system.	mount(2)
	mount, umount: mount and dismount file system. ..	mount(8)
	mount, umount: mount or remove file system.	mount(2)
mtab:	mounted file system table.	mtab(5)
graphics openpl, erase, label, line, circle, arc,	move, cont, point, linemod, space, closepl:	plot(3X)
mv:	move or rename files.	mv(1)
lseek:	move read/write pointer.	lseek(2)
	ms: text formatting macros.	ms(7)
	msgs: system messages and junk mail program.	msgs(1)
	mt: magnetic tape manipulating program.	mt(1)
	mtab: mounted file system table.	mtab(5)
	mtio: tape device files.	mtio(4)
eyacc: modified yacc allowing	much improved error recovery.	eyacc(1)
select: synchronous I/O	multiplexing.	select(2)
fsplit: split a	multi-routine FORTRAN file into individual files.	fsplit(1)
switch:	multi-way command branch.	csh(1)
	mv: move or rename files.	mv(1)
from: who is	my mail from?.	from(1)

getlogin: get log-in	name.....getlogin(3)
getsockname: get socket	name.....getsockname(2)
pwd: working directory	name.....pwd(1)
tty: get terminal	name.....tty(1)
hosts: host	name database.....hosts(5)
networks: network	name database.....networks(5)
protocols: protocol	name database.....protocols(5)
nm: print	name list.....nm(1)
cvtumap: convert name trees from SR8 to SR9	name mapping.....cvtumap(8)
rename: change the	name of a file.....rename(2)
ttyname, isatty: find	name of a terminal.....ttyname(3)
getpeername: get	name of connected peer.....getpeername(2)
gethostname, sethostname: get/set	name of current host.....gethostname(2)
hostname: set or print	name of current host system.....hostname(1)
bind: bind a	name to a socket.....bind(2)
cvtumap: convert	name trees from SR8 to SR9 name mapping.....cvtumap(8)
foreach: loop over list of	names.....csh(1)
term: conventional	names for terminals.....term(7)
checkeq: check files that use eqn(1) or	neqn(1).....checkeq(1)
ntohl, ntohs: convert values between host and	netstat: show network status.....netstat(1)
getnetbyname, setnetent, endnetent: get	network byte order. htonl, htons,.....byteorder(3n)
gethostbyname, sethostent, endhostent: get	network entry. getnetent, getnetbyaddr,.....getnetent(3n)
ifconfig: configure	network host entry. gethostent, gethostbyaddr,.....gethostent(3n)
intro: introduction to	network interface parameters.....ifconfig(8C)
networks:	network library functions.....intro(3n)
routed:	network name database.....networks(5)
netstat: show	network routing daemon.....routed(8C)
networking: introduction to	network status.....netstat(1)
	networking facilities.....intro(4N)
	networking: introduction to networking facilities.....intro(4N)
	networks: network name database.....networks(5)
open a file for reading or writing, or create a	new file. open:.....open(2)
arcv: convert archive files to	new format.....arcv(8)
fork: create a	new process.....fork(2)
vfork: spawn a	new process in a more efficient way.....vfork(2)
login: login	new user.....csh(1)
aliases file.	newaliases: rebuild the database for the mail.....newaliases(1)
dbminit, fetch, store, delete, firstkey,	nextkey: database subroutines.....dbm(3X)
gettable: get	NIC format host tables from a host.....gettable(8C)
htable: convert	NIC standard format host tables.....htable(8)
	nice, nohup: run a command at a different priority nice(1)
	nice: run low priority process.....csh(1)
	nm: print name list.....nm(1)
wall: write to all users on a	node.....wall(1)
uptime: show how long a	node has been up.....uptime(1)
update auxiliary system administrator's	nodes. update_slave:.....update_slave(8)
flush_cache - clear the	node's acl_cache.....flush_cache(8)
nice,	nohup: run a command at a different priority.....nice(1)
setjmp, longjmp:	nohup: run command immune to hangsups.....csh(1)
	non-local goto.....setjmp(3)

notify: request immediate	notification.....csh(1)
soelim: eliminate .so's from	notify: request immediate notification.....csh(1)
tbl: format tables for	nrff input.....soelim(1)
colcrt: filter	nrff or troff.....tbl(1)
deroff: remove	nrff output for CRT previewing.....colcrt(1)
checknr: check	nrff: text formatting.....nrff(1)
network byte order. htonl, htons,	nrff, troff, tbl, and eqn constructs.....deroff(1)
order. htonl, htons, ntohl,	nrff/troff files.....checknr(1)
	ntohl, ntohs: convert values between host and.....byteorder(3n)
	ntohs: convert values between host and network byteorder(3n)
	null: data sink.....null(4)
phones: remote host phone	number: convert Arabic numerals to English.....number(6)
arithmetic: provide drill in	number database.....phones(5)
random, srandom, initstate, setstate: better random	number facts.....arithmetic(6)
strip: strip symbol and line	number generator and associated routines.....random(3)
atof, atoi, atol: convert ASCII to	number information from an object file.....strip(1)
intro: introduction to system calls and error	numbers.....atof(3)
number: convert Arabic	numbers.....intro(2)
size: size of an	numerals to English.....number(6)
strings: find the printable strings in an	object file.....size(1)
strip symbol and line number information from an	object file.....strings(1)
lorder: find ordering relation for an	object file. strip:.....strip(1)
od:	object library.....lorder(1)
	octal, decimal, hex, ASCII dump.....od(1)
prmail: print out mail in the post	od: octal, decimal, hex, ASCII dump.....od(1)
lpr: print files	office.....prmail(1)
login: sign	off-line.....lpr(1)
crypt, encrypt: a	on.....login(1)
	one-way hashing encryption algorithm.....crypt(3)
nohup: run a command at a different priority	onintr: process interrupts in command scripts.....csh(1)
a program file, including aliases and paths	nice.....nice(1)
file. open:	which: locate.....which(1)
fopen, freopen, fdopen:	open a file for reading or writing, or create a new.....open(2)
flock: place or remove an advisory lock on an	open a stream.....fopen(3S)
a new file.	open file.....flock(2)
closedir: directory operations.	open: open a file for reading or writing, or create.....open(2)
cont, point, linemod, space, closepl: graphics	opendir, readdir, telldir, seekdir, rewinddir,.....directory(3)
tgetstr, tgoto, tputs: terminal independent	openpl, erase, label, line, circle, arc, move,.....plot(3X)
bcopy, bcmp, bzero, ffs: bit and byte string	operation routines. tgetent, tgetnum, tgetflag,.....termcap(3X)
telldir, seekdir, rewinddir, closedir: directory	operations.....bstring(3)
strcpy, strncpy, strlen, index, rindex: string	operations. opendir, readdir,.....directory(3)
join: relational database	operations. strcat, strncat, strcmp, strncmp,.....string(3)
curses: screen functions with	operator.....join(1)
stty: set terminal	optimized cursor motion.....curses(3X)
getsockopt, setsockopt: get/set	options.....stty(1)
ntohs: convert values between host and network byte	options on sockets.....getsockopt(2)
lorder: find	order. htonl, htons, ntohl,.....byteorder(3n)
a.out: cc	ordering relation for an object library.....lorder(1)
terminate a process after flushing any pending	output.....a.out(5)
	output. exit:.....exit(3)

ecvt, fcvt, gcvt:	output conversion.	ecvt(3)
printf, fprintf, sprintf: formatted	output conversion.	printf(3S)
fold: fold long lines for finite width	output device.	fold(1)
colcrt: filter nroff	output for CRT previewing.	colcrt(1)
vprintf, fprintf, vsprintf: print formatted	output of a varargs argument list.	vprintf(3S)
foreach: loop	over list of names.	csh(1)
sendmail: send mail	over the internet.	sendmail(8)
exec:	overlay shell with specified command.	csh(1)
chown: change the	owner of files.	chown(8)
chown: change	owner or group of a file.	chown(2)
	pac: printer/plotter accounting information.	pac(8)
stdio: standard buffered input/output	package.	intro(3S)
more,	page: file perusal filter for CRT viewing.	more(1)
more,	page: file perusal filter for CRT viewing.	page(1)
getpagesize: get system	page size.	getpagesize(2)
pagesize: print system	page size.	pagesize(1)
intro: miscellaneous useful information	pages.	intro(7)
man: macros for formatting manual	pages.	man(7)
	pagesize: print system page size.	pagesize(1)
socketpair: create a	pair of connected sockets.	socketpair(2)
me: macros for formatting	papers.	me(7)
ifconfig: configure network interface	parameters.	ifconfig(8C)
	passwd: change log-in password.	passwd(1)
	passwd: password file.	passwd(5)
getpass: read a	password.	getpass(3)
passwd: change log-in	password.	passwd(1)
crpasswd: create	password and group files.	crpasswd(8)
passwd:	password file.	passwd(5)
getpwuid, getpwnam, setpwent, endpwent: get	password file entry. getpwent,	getpwent(3)
getwd: get current working directory	pathname.	getwd(3)
which: locate a program file, including aliases and	paths.	which(1)
grep, egrep, fgrep: search a file for a	pattern.	grep(1)
awk:	pattern scanning and processing language.	awk(1)
	pause: stop until signal.	pause(3C)
popen,	pclose: initiate I/O to and from a process.	popen(3)
getpeername: get name of connected	peer.	getpeername(2)
exit: terminate a process after flushing any	pending output.	exit(3)
update: update the super-block	periodically.	update(8)
mesg:	permit or deny messages.	mesg(1)
ptx:	permuted index.	ptx(1)
limit: alter	per-process resource limitations.	csh(1)
messages.	perror, sys_errlist, sys_ner: system error.	perror(3)
more, page: file	perusal filter for CRT viewing.	more(1)
more, page: file	perusal filter for CRT viewing.	page(1)
phones: remote host	phone number database.	phones(5)
	phones: remote host phone number database.	phones(5)
split: split a file into	pieces.	split(1)
	pipe: create an interprocess communication channel.	pipe(2)
tee:	pipe fitting.	tee(1)
bg:	place job in background.	csh(1)

	flock:	place or remove an advisory lock on an open file.	flock(2)
	fish:	play "Go Fish"	fish(6)
	mille:	play Mille Boumes	mille(6)
	worm:	Play the growing worm game.	worm(6)
	plot:	graphics filters.	plot(1G)
	plot:	graphics interface.	plot(5)
erase, label, line, circle, arc, move, cont,	point, linemod, space, closepl:	graphics interface	plot(3X)
lseek: move read/write	pointer.		lseek(2)
popd:	pop shell directory stack.		csh(1)
	popd: pop shell directory stack.		csh(1)
	popen, pclose:	initiate I/O to and from a process.	popen(3)
prmail: print out mail in the	post office.		prmail(1)
root. exp, log, log10,	pow, sqrt:	exponential, logarithm, power, square	exp(3M)
exp, log, log10, pow, sqrt:	power, square root.		exp(3M)
	pr:	print file.	pr(1)
	pr:	pr to the line printer.	print(1)
colcrt: filter nroff output for CRT	previewing.		colcrt(1)
unget: undo a	previous get of an SCCS file.		unget(1)
types:	primitive system data types.		types(5)
cat: catenate and	print.		cat(1)
fortune:	print a random adage.		fortune(6)
prs:	print an SCCS file.		prs(1)
cal:	print calendar.		cal(1)
hashstat:	print command hashing statistics.		csh(1)
jobs:	print current job list.		csh(1)
sact:	print current SCCS file editing activity.		sact(1)
whoami:	print effective current user ID.		whoami(1)
pr:	print file.		pr(1)
lpr:	print files off-line.		lpr(1)
vprintf, vfprintf, vsprintf:	print formatted output of a varargs argument list.		vprintf(3S)
fpr:	print FORTRAN file.		fpr(1)
history:	print history event list.		csh(1)
hostid: set or	print identifier of current host system.		hostid(1)
banner:	print large banner on printer.		banner(6)
nm:	print name list.		nm(1)
hostname: set or	print name of current host system.		hostname(1)
prmail:	print out mail in the post office.		prmail(1)
printenv:	print out the environment.		printenv(1)
	print: pr to the line printer.		print(1)
pagesize:	print system page size.		pagesize(1)
date:	print the date.		date(1)
diction, explain:	print wordy sentences; thesaurus for diction.		diction(1)
strings: find the	printable strings in an object file.		strings(1)
	printcap: printer capability data base.		printcap(5)
	printenv: print out the environment.		printenv(1)
banner: print large banner on	printer.		banner(6)
print: pr to the line	printer.		print(1)
printcap:	printer capability data base.		printcap(5)
lpc: line	printer control program.		lpc(8)
lpd: line	printer daemon.		lpd(8)

lprm: remove jobs from the line	printer spooling queue.lprm(1)
pac: printer/plotter accounting information.....pac(8)	
conversion. printf, fprintf, sprintf: formatted outputprintf(3S)	
setpriority: get/set program scheduling	priority. getpriority,.....getpriority(2)
renice: alter	priority of running processesrenice(8)
nice: run low	priority processcsh(1)
nice, nohup: run a command at a different	priority.nice(1)
nice: run low priority	prmail: print out mail in the post office.prmail(1)
stop: halt a job or	process.csh(1)
_exit: terminate a	process.csh(1)
fork: create a new	process.exit(2)
kill: terminate a specified	process.fork(2)
kill: send signal to a	process.kill(1)
popen, pclose: initiate I/O to and from a	process.kill(2)
wait: await completion of	process.popen(3)
exit: terminate a	process.wait(1)
getpggrp: get	process after flushing any pending output.exit(3)
killpg: send signal to a	process group.....getpggrp(2)
setpggrp: set	process group.....killpg(2)
getpid, getppid: get	process group.....setpggrp(2)
vfork: spawn a new	process identification.....getpid(2)
onintr:	process in a more efficient way.....vfork(2)
ps:	process interrupts in command scripts.csh(1)
times: get	process status.....ps(1)
wait, wait3: wait for	process times.times(3C)
ptrace:	process to terminate.wait(2)
kill: kill jobs and	process trace.ptrace(2)
renice: alter priority of running	processes.csh(1)
wait: wait for background	processes.....renice(8)
awk: pattern scanning and	processes to complete.csh(1)
halt: stop the	processing language.awk(1)
m4: macro	processor.....halt(8)
reboot: reboot system or halt	processor.....m4(1)
reboot: reboot the	processor.....reboot(2)
end, etext, edata: last location in	processor.....reboot(8)
ftp: file transfer	program.....end(3)
lpc: line printer control	program.....ftp(1C)
lpq: spool queue examination	program.....lpc(8)
msgs: system messages and junk mail	program.....lpq(1)
mt: magnetic tape manipulating	program.....msgs(1)
talkd: server for talk(1)	program.....mt(1)
units: conversion	program.....talkd(8C)
whereis: locate binary and/or manual for	program.....units(1)
writed: daemon for write(1)	program.....whereis(1)
cb: C	program.....writed(8C)
which: locate a	program beautifier.cb(1)
make: maintain	program file, including aliases and paths (csh.....which(1)
getpriority, setpriority: get/set	program groups.make(1)
indent: indent and format C	program scheduling priority.....getpriority(2)
	program source.....indent(1)

assert:	program verification.....	assert(3X)
lint: a C	program verifier.....	lint(1)
lex: generator of lexical analysis	programs.....	lex(1)
xstr: extract strings from C	programs to implement shared strings.....	xstr(1)
sup: set UNIX-style	protection.....	sup(8)
default_acl: change default file	protection environment.....	default_acl(2)
arp: Address Resolution	Protocol.....	arp(4P)
tcp: Internet Transmission Control	Protocol.....	tcp(4P)
telnet: user interface to the TELNET	protocol.....	telnet(1C)
udp: Internet User Datagram	Protocol.....	udp(4P)
getprotobyname, setprotoent, endprotoent: get	protocol entry. getprotoent, getprotobyname,	getprotoent(3n)
inet: Internet	protocol family.....	inet(4F)
protocols:	protocol name database.....	protocols(5)
ftpd: DARPA Internet File Transfer	Protocol server.....	ftpd(8C)
telnetd: DARPA TELNET	protocol server.....	telnetd(8C)
tftpd: DARPA Trivial File Transfer	Protocol server.....	tftpd(8C)
	protocols: protocol name database.....	protocols(5)
arithmetic:	provide drill in number facts.....	arithmetic(6)
false, true:	provide truth values.....	false(1)
true, false:	provide truth values.....	true(1)
	prs: print an SCCS file.....	prs(1)
	ps: process status.....	ps(1)
pty:	pseudo terminal driver.....	pty(4)
	psignal, sys_siglist: system signal messages.....	psignal(3)
crpty: create	psuedo tty device entries.....	crpty(8)
	ptrace: process trace.....	ptrace(2)
	ptx: permuted index.....	ptx(1)
	pty: pseudo terminal driver.....	pty(4)
tar: tape (and general	purpose) archiver.....	tar(1)
ungetc:	push character back into input stream.....	ungetc(3S)
pushd:	push shell directory stack.....	csh(1)
	pushd: push shell directory stack.....	csh(1)
puts, fputs:	put a string on a stream.....	puts(3S)
putc, putchar, fputc, putw:	put character or word on a stream.....	putc(3S)
on a stream.	putc, putchar, fputc, putw: put character or word...	putc(3S)
stream. putc,	putchar, fputc, putw: put character or word on a....	putc(3S)
	puts, fputs: put a string on a stream.....	puts(3S)
putc, putchar, fputc,	putw: put character or word on a stream.....	putc(3S)
	pwd: working directory name.....	pwd(1)
	qsort: quicker sort.....	qsort(3)
insque, remque: insert or remove an element in a	queue.....	insque(3)
lprm: remove jobs from the line printer spooling	queue.....	lprm(1)
lpq: spool	queue examination program.....	lpq(1)
qsort:	quicker sort.....	qsort(3)
	rain: animated raindrops display.....	rain(6)
rain: animated	raindrops display.....	rain(6)
fortune: print a	random adage.....	fortune(6)
ranlib: convert archives to	random libraries.....	ranlib(1)
random, srandom, initstate, setstate: better	random number generator and associated routines.....	random(3)
number generator and associated routines.	random, srandom, initstate, setstate: better random	random(3)

	ranlib: convert archives to random libraries.	ranlib(1)
	ratfor: rational FORTRAN dialect.	ratfor(1)
ratfor:	rational FORTRAN dialect.	ratfor(1)
	rc: boot time shell script.	rc(8)
stream to a remote command.	rcmd, resvport, ruserok: routines for returning a ...	rcmd(3X)
	rcp: remote file copy.	rcp(1C)
getpass:	read a password.	getpass(3)
source:	read commands from file.	csh(1)
read, readv:	read input.	read(2)
	read, readv: read input.	read(2)
readlink:	read value of a symbolic link.	readlink(2)
directory operations. opendir,	readdir, telldir, seekdir, rewinddir, closedir:	directory(3)
open: open a file for	reading or writing, or create a new file.	open(2)
	readlink: read value of a symbolic link.	readlink(2)
read,	readv: read input.	read(2)
lseek: move	read/write pointer.	lseek(2)
setregid: set	real and effective group ID.	setregid(2)
setreuid: set	real and effective user ID.	setreuid(2)
malloc, free,	realloc, calloc, alloca: memory allocator.	malloc(3)
swapul:	rearrange underlining.	swapul(8)
	reboot: reboot system or halt processor.	reboot(2)
	reboot: reboot the processor.	reboot(8)
reboot:	reboot system or halt processor.	reboot(2)
reboot:	reboot the processor.	reboot(8)
newaliases:	rebuild the database for the mail aliases file.	newaliases(1)
recv, recvfrom, recvmsg:	receive a message from a socket.	recv(2)
mail: send and	receive mail.	mail(1)
binmail: send or	receive mail among users.	binmail(1)
rmail: handle remote mail	received via uucp.	rmail(1)
	re_comp, re_exec: regular expression handler.	regex(3)
rehash:	recompute command hash table.	csh(1)
eyacc: modified yacc allowing much improved error	recovery.	eyacc(1)
socket.	recv, recvfrom, recvmsg: receive a message from arecv(2)	
recv,	recvfrom, recvmsg: receive a message from a socket.	recv(2)
recv, recvfrom,	recvmsg: receive a message from a socket.	recv(2)
eval:	re-evaluate shell data.	csh(1)
re_comp,	re_exec: regular expression handler.	regex(3)
documents.	refer: find and insert literature references in	refer(1)
man: display	reference manual information.	man(1)
man: display	reference manual information.	man.1.11(12)
build inverted index for a bibliography; find	references in a bibliography. indxbib, lookbib:	lookbib(1)
refer: find and insert literature	references in documents.	refer(1)
re_comp, re_exec:	regular expression handler.	regex(3)
	rehash: recompute command hash table.	csh(1)
comm: select or	reject lines common to two sorted files.	comm(1)
lorder: find ordering	relation for an object library.	lorder(1)
join:	relational database operator.	join(1)
sigpause: atomically	release blocked signals and wait for interrupt.	sigpause(2)
leave:	remind you when you have to leave.	leave(1)
calendar:	reminder service.	calendar(1)

ruserok: routines for returning a stream to a	remote command. rcmd, resvport,	rcmd(3X)
rexec: return stream to a	remote command.	rexec(3X)
rexecd: remote execution server	rexecd(8C)	
rcp: remote file copy.	rcp(1C)	
uucsend: send a file to a	remote host.	uucsend(1C)
remote: remote host description file.	remote(5)	
phones: remote host phone number database.	phones(5)	
rlogin: remote log-in.	rlogin(1C)	
rlogind: remote log-in server	rlogind(8C)	
rmail: handle	remote mail received via uucp.	rmail(1)
remote: remote host description file	remote(5)	
rsh: remote Shell.	rsh(1C)	
rshd: remote Shell server	rshd(8C)	
tip, cu: connect to a	remote system.	cu(1C)
tip, cu: connect to a	remote system.	tip(1C)
rmel: remove a delta from an SCCS file.	rmel(1)	
rmdir: remove a directory file.	rmdir(2)	
unalias: remove aliases.	csh(1)	
flock: place or	remove an advisory lock on an open file.	flock(2)
insque, remque: insert or	remove an element in a queue.	insque(3)
colrm: remove columns from a file.	colrm(1)	
unlink: remove directory entry.	unlink(2)	
unsetenv: remove environment variables.	csh(1)	
mount, umount: mount or	remove file system.	mount(2)
lprm: remove jobs from the line printer spooling queue.	lprm(1)	
deroff: remove nroff, troff, tbl, and eqn constructs.	deroff(1)	
unlimit: remove resource limitations.	csh(1)	
rm, rmdir: remove (unlink) directories or files.	rm(1)	
insque, remque: insert or remove an element in a queue.	insque(3)	
rename: change the name of a file.	rename(2)	
mv: move or	rename files.	mv(1)
renice: alter priority of running processes	renice(8)	
fix_cache -	repair acl cache hash chains.	fix_cache(8)
while: repeat commands conditionally.	csh(1)	
repeat: execute command repeatedly.	csh(1)	
uniq: report	repeated lines in a file.	uniq(1)
repeat: execute command	repeatedly.	csh(1)
yes: be	repetitively affirmative.	yes(1)
uniq: report repeated lines in a file.	uniq(1)	
fseek, ftell, rewind: reposition a stream.	fseek(3S)	
notify: request immediate notification.	csh(1)	
reset: reset the teletype bits to a sensible state.	reset(1)	
reset: reset the teletype bits to a sensible state.	reset(1)	
arp: Address	Resolution Protocol.	arp(4P)
getrlimit: control maximum system	resource consumption.	getrlimit(2)
limit: alter per-process	resource limitations.	csh(1)
unlimit: remove	resource limitations.	csh(1)
getrusage: get information about	resource utilization.	getrusage(2)
suspend: suspend a shell,	resuming its superior.	csh(1)
rexec: return stream to a remote command.	rexec(3X)	

rcmd, rresvport, ruserok: routines for	returning a stream to a remote command.....rcmd(3X)
col: filter	rev: reverse lines of a file.....rev(1)
rev:	reverse line feeds.....col(1)
fseek, ftell,	reverse lines of a file.....rev(1)
opendir, readdir, telldir, seekdir,	rewind: reposition a stream.....fseek(3S)
	rewinddir, closedir: directory operations.....directory(3)
	rexec: return stream to a remote command.....rexec(3X)
	rexecd: remote execution server.....rexecd(8C)
strcmp, strncmp, strcpy, strncpy, strlen, index,	rindex: string operations. strcat, strncat.....string(3)
	rlogin: remote log-in.....rlogin(1C)
	rlogind: remote log-in server.....rlogind(8C)
	rm, rmdir: remove (unlink) directories or files.....rm(1)
	rmail: handle remote mail received via uucp.....rmail(1)
	rmdel: remove a delta from an SCCS file.....rmdel(1)
	rmdir: remove a directory file.....rmdir(2)
rm,	rmdir: remove (unlink) directories or files.....rm(1)
	roffbib: run off bibliographic database.....roffbib(1)
pow, sqrt: exponential, logarithm, power, square	root. exp, log, log10,exp(3M)
addroot: add a	root ID.....addroot(8)
	route: manually manipulate the routing tables.....route(8C)
	routed: network routing daemon.....routed(8C)
inet_netof: Internet address manipulation	routines. inet_ntoa, inet_makeaddr, inet_lnaof,.....inet(3n)
better random number generator and associated	routines. random, srandom, initstate, setstate:.....random(3)
tgoto, tputs: terminal independent operation	routines. tgetent, tgetnum, tgetflag, tgetstr,termcap(3X)
command. rcmd, rresvport, ruserok:	routines for returning a stream to a remote.....rcmd(3X)
routed: network	routing daemon.....routed(8C)
route: manually manipulate the	routing tables.....route(8C)
to a remote command. rcmd,	rresvport, ruserok: routines for returning a stream.....rcmd(3X)
	rsh: remote Shell.....rsh(1C)
	rshd: remote Shell server.....rshd(8C)
nice, nohup:	run a command at a different priority.....nice(1)
nohup:	run command immune to hangups.....csh(1)
nice:	run low priority process.....csh(1)
roffbib:	run off bibliographic database.....roffbib(1)
renice: alter priority of	running processes.....renice(8)
	uptime: show host status of local machines.....uptime(1C)
remote command. rcmd, rresvport,	ruserok: routines for returning a stream to a.....rcmd(3X)
	rwwho: who's logged in on local machines.....rwho(1C)
	rwhod: system status server.....rwhod(8C)
	sact: print current SCCS file editing activity.....sact(1)
brk,	sbrk: change data segment size.....brk(2)
scandir:	scan a directory.....scandir(3)
	scandir: scan a directory.....scandir(3)
	scanf, fscanf, sscanf: formatted input conversion.....scanf(3S)
awk: pattern	scanning and processing language.....awk(1)
cdc: change the delta commentary of an	SCCS delta.....cdc(1)
comb: combine	SCCS deltas.....comb(1)
delta: make a delta (change) to an	SCCS file.....delta(1)
get: get a version of an	SCCS file.....get(1)
prs: print an	SCCS file.....prs(1)

rmidel: remove a delta from an	SCCS file.....rmidel(1)
sccsdiff: compare two versions of an	SCCS file.....sccsdiff(1)
sccsfile: format of Source Code Control System	(SCCS) file.....sccsfile(5)
unget: undo a previous get of an	SCCS file.....unget(1)
val: validate	SCCS file.....val(1)
sact: print current	SCCS file editing activity.....sact(1)
admin: create and administer	SCCS files.....admin(1)
what: identify	SCCS files.....what(1)
(SCCS) file	sccsdiff: compare two versions of an SCCS file.....sccsdiff(1)
getpriority, setpriority: get/set program	sccsfile: format of Source Code Control System.....sccsfile(5)
clear: clear terminal	scheduling priority.....getpriority(2)
curses:	screen.....clear(1)
ex. vi:	screen functions with optimized cursor motion.....curses(3X)
rc: boot time shell	screen-oriented (visual) display editor based on.....vi(1)
onintr: process interrupts in command	script.....rc(8)
grep, egrep, fgrep:	script: make typescript of a terminal session.....script(1)
opendir, readdir, telldir,	scripts.....csh(1)
brk, sbrk: change data	search a file for a pattern.....grep(1)
comm:	sed: stream editor.....sed(1)
case:	seekdir, rewinddir, closedir: directory operations.....directory(3)
uusend:	segment size.....brk(2)
send, sendto, sendmsg:	select or reject lines common to two sorted files.....comm(1)
mail:	select: synchronous I/O multiplexing.....select(2)
sendmail:	selector in switch.....csh(1)
binmail:	send a file to a remote host.....uusend(1C)
socket:	send a message from a socket.....send(2)
kill:	send and receive mail.....mail(1)
killpg:	send mail over the internet.....sendmail(8)
aliases: aliases file for	send or receive mail among users.....binmail(1)
send, sendto,	send, sendto, sendmsg: send a message from a.....send(2)
send,	send signal to a process.....kill(2)
reset: reset the teletype bits to a	send signal to a process group.....killpg(2)
diction, explain: print wordy	sendmail.....aliases(5)
ftpd: DARPA Internet File Transfer Protocol	sendmail: send mail over the internet.....sendmail(8)
rexecd: remote execution	sendmsg: send a message from a socket.....send(2)
rlogind: remote log-in	sendto, sendmsg: send a message from a socket.....send(2)
rshd: remote Shell	sensible state.....reset(1)
rwhod: system status	sentences; thesaurus for diction.....diction(1)
telnetd: DARPA TELNET protocol	server.....ftpd(8C)
tftpd: DARPA Trivial File Transfer Protocol	server.....rexecd(8C)
talkd:	server.....rlogind(8C)
calendar: reminder	server.....rshd(8C)
services: database of Internet	server.....rwhod(8C)
logout: end	server.....telnetd(8C)
	server.....tftpd(8C)
	server for talk(1) program.....talkd(8C)
	service.....calendar(1)
	services.....services(5)
	services: database of Internet services.....services(5)
	session.....csh(1)

script: make typescript of a terminal	session.....script(1)
ascii: map of ASCII character	set.....ascii(7)
sigstack:	set and/or get signal stack context.....sigstack(2)
	set: change value of shell variable.....csh(1)
sigsetmask:	set current signal mask.....sigsetmask(2)
utimes:	set file times.....utimes(2)
setgroups:	set group access list.....setgroups(2)
apply: apply a command to a	set of arguments.....apply(1)
hostid:	set or print identifier of current host system.....hostid(1)
hostname:	set or print name of current host system.....hostname(1)
setpgrp:	set process group.....setpgrp(2)
setregid:	set real and effective group ID.....setregid(2)
setreuid:	set real and effective user ID.....setreuid(2)
stty:	set terminal options.....stty(1)
tabs:	set terminal tabs.....tabs(1)
sup:	set UNIX-style protection.....sup(8)
setuid seteuid setruid setgid setegid setrgid:	set user and group ID.....net(3n)
setuid, seteuid, setruid, setgid, setegid, setrgid:	set user and group ID.....setuid(3)
setenv:	set variable in environment.....csh(1)
a stream.	setbuf, setbuffer, setlinebuf: assign buffering to.....setbuf(3S)
stream. setbuf,	setbuffer, setlinebuf: assign buffering to a.....setbuf(3S)
setuid seteuid setruid setgid	setgid setrgid: set user and group ID.....net(3n)
setuid, seteuid, setruid, setgid,	setgid, setrgid: set user and group ID.....setuid(3)
	setenv: set variable in environment.....csh(1)
and group ID setuid	seteuid setruid setgid setegid setrgid: set user.....net(3n)
user and group ID. setuid,	seteuid, setruid, setgid, setegid, setrgid: set.....setuid(3)
umask:	set/get file creation mask.....umask(2)
setuid seteuid setruid	setgid setegid setrgid: set user and group ID.....net(3n)
setuid, seteuid, setruid,	setgid, setegid, setrgid: set user and group ID.....setuid(3)
getgrent, getgrgid, getgrnam,	setgrent, endgrent: get group file entry.....getgrent(3)
	setgroups: set group access list.....setgroups(2)
gethostent, gethostbyaddr, gethostbyname,	sethostent, endhostent: get network host entry.....gethostent(3n)
host. gethostid,	sethostid: get/set unique identifier of current.....gethostid(2)
gethostname,	sethostname: get/set name of current host.....gethostname(2)
getitimer,	setitimer: get/set value of interval timer.....getitimer(2)
	setjmp, longjmp: non-local goto.....setjmp(3)
setbuf, setbuffer,	setlinebuf: assign buffering to a stream.....setbuf(3S)
getnetent, getnetbyaddr, getnetbyname,	setnetent, endnetent: get network entry.....getnetent(3n)
	setpgrp: set process group.....setpgrp(2)
getpriority,	setpriority: get/set program scheduling priority.....getpriority(2)
getprotoent, getprotobyname, getprotobynumber,	setprotoent, endprotoent: get protocol entry.....getprotoent(3n)
getpwent, getpwuid, getpwnam,	setpwent, endpwent: get password file entry.....getpwent(3)
	setregid: set real and effective group ID.....setregid(2)
setuid seteuid setruid setgid setegid	setreuid: set real and effective user ID.....setreuid(2)
setuid, seteuid, setruid, setgid, setegid,	setrgid: set user and group ID.....net(3n)
ID setuid seteuid	setrgid: set user and group ID.....setuid(3)
group ID. setuid, seteuid,	setruid setgid setegid setrgid: set user and group.....net(3n)
getservent, getservbyport, getservbyname,	setruid, setgid, setegid, setrgid: set user and.....setuid(3)
getsockopt,	setservent, endservent: get service entry.....getservent(3n)
	setsockopt: get/set options on sockets.....getsockopt(2)

associated routines.	random, srandom, initstate,	setstate: better random number generator and.....random(3)
	gettimeofday,	settimeofday: get/set date and time.....gettimeofday(2)
	user and group ID	setuid seteuid setruid setgid setegid setrgid: set.....net(3n)
	set user and group ID.	setuid, seteuid, setruid, setgid, setegid, setrgid:.....setuid(3)
		sh: command language.....sh(1)
nice, nohup: run a command at a different priority	nice(1)
xstr: extract strings from C programs to implement	shared strings.....	xstr(1)
exit: leave	shell.....	csh(1)
rsh: remote	Shell.....	rsh(1C)
cp /bin/start_csh: start a C	shell.....	start_csh(1)
cp /bin/start_sh: start a Bourne	Shell.....	start_sh(1)
system: issue a	shell command.....	system(3)
csh: a	shell (command interpreter) with C-like syntax.....	csh(1)
ver: change the version of	Shell commands.....	ver(8)
eval: re-evaluate	shell data.....	csh(1)
popd: pop	shell directory stack.....	csh(1)
pushd: push	shell directory stack.....	csh(1)
alias:	shell macros.....	csh(1)
suspend: suspend a	shell, resuming its superior.....	csh(1)
rc: boot time	shell script.....	rc(8)
rshd: remote	Shell server.....	rshd(8C)
set: change value of	shell variable.....	csh(1)
@: arithmetic on	shell variables.....	csh(1)
unset: discard	shell variables.....	csh(1)
exec: overlay	shell with specified command.....	csh(1)
	shift: manipulate argument list.....	csh(1)
groups:	show group memberships.....	groups(1)
ruptime:	show host status of local machines.....	ruptime(1C)
uptime:	show how long a node has been up.....	uptime(1)
netstat:	show network status.....	netstat(1)
uusnap:	show snapshot of the UUCP system.....	uusnap(8C)
shutdown:	shut down part of a full-duplex socket connection.....	shutdown(2)
connection.	shutdown: shut down part of a full-duplex socket ..	shutdown(2)
	sigblock: block signals.....	sigblock(2)
login:	sign on.....	login(1)
pause: stop until	signal.....	pause(3C)
signal: simplified software	signal facilities.....	signal(3C)
sigvec: software	signal facilities.....	sigvec(2)
sigsetmask: set current	signal mask.....	sigsetmask(2)
psignal, sys_siglist: system	signal messages.....	psignal(3)
	signal: simplified software signal facilities.....	signal(3C)
sigstack: set and/or get	signal stack context.....	sigstack(2)
kill: send	signal to a process.....	kill(2)
killpg: send	signal to a process group.....	killpg(2)
sigblock: block	signals.....	sigblock(2)
sigpause: atomically release blocked	signals and wait for interrupt.....	sigpause(2)
wait for interrupt.	sigpause: atomically release blocked signals and....	sigpause(2)
	sigsetmask: set current signal mask.....	sigsetmask(2)
	sigstack: set and/or get signal stack context.....	sigstack(2)
	sigvec: software signal facilities.....	sigvec(2)

signal:	simplified software signal facilities.....	signal(3C)
trigonometric functions.	sin, cos, tan, asin, acos, atan, atan2:	sin(3M)
	sinh, cosh, tanh: hyperbolic functions.....	sinh(3M)
null: data	sink.....	null(4)
brk, sbrk: change data segment	size.....	brk(2)
getdtablesize: get descriptor table	size.....	getdtablesize(2)
getpagesize: get system page	size.....	getpagesize(2)
pagesize: print system page	size.....	pagesize(1)
size:	size of an object file.....	size(1)
	size: size of an object file.....	size(1)
	sleep: suspend execution for an interval.....	sleep(1)
	sleep: suspend execution for interval.....	sleep(3)
spline: interpolate	smooth curve.....	spline(1G)
uusnap: show	snapshot of the UUCP system.....	uusnap(8C)
accept: accept a connection on a	socket.....	accept(2)
bind: bind a name to a	socket.....	bind(2)
connect: initiate a connection on a	socket.....	connect(2)
listen: listen for connections on a	socket.....	listen(2)
recv, recvfrom, recvmsg: receive a message from a	socket.....	recv(2)
send, sendto, sendmsg: send a message from a	socket.....	send(2)
shutdown: shut down part of a full-duplex	socket connection.....	shutdown(2)
	socket: create an endpoint for communication.....	socket(2)
getsockname: get	socket name.....	getsockname(2)
	socketpair: create a pair of connected sockets.....	socketpair(2)
getsockopt, setsockopt: get/set options on	sockets.....	getsockopt(2)
socketpair: create a pair of connected	sockets.....	socketpair(2)
	soelim: eliminate .so's from nroff input.....	soelim(1)
soft_link, soft_unlink: create or delete	soft links.....	soft_link(2)
links.	soft_link, soft_unlink: create or delete soft.....	soft_link(2)
soft_link,	soft_unlink: create or delete soft links.....	soft_link(2)
signal: simplified	software signal facilities.....	signal(3C)
sigvec:	software signal facilities.....	sigvec(2)
qsort: quicker	sort.....	qsort(3)
tsort: topological	sort.....	tsort(1)
sortbib:	sort bibliographic database.....	sortbib(1)
sort:	sort or merge files.....	sort(1)
	sort: sort or merge files.....	sort(1)
	sortbib: sort bibliographic database.....	sortbib(1)
comm: select or reject lines common to two	sorted files.....	comm(1)
look: find lines in a	sorted list.....	look(1)
soelim: eliminate	.so's from nroff input.....	soelim(1)
indent: indent and format C program	source.....	indent(1)
mkstr: create an error message file by massaging C	source.....	mkstr(1)
sccsfile: format of	Source Code Control System (SCCS) file.....	sccsfile(5)
	source: read commands from file.....	csh(1)
line, circle, arc, move, cont, point, linemod,	space, closepl: graphics interface. erase, label,.....	plot(3X)
expand, unexpand: expand tabs to	spaces and vice versa.....	expand(1)
vfork:	spawn a new process in a more efficient way.....	vfork(2)
exec: overlay shell with	specified command.....	csh(1)
truncate: truncate a file to a	specified length.....	truncate(2)

kill: terminate a	specified process.....	kill(1)
spell, spellin, spellout: find	spelling errors.	spell(1)
spell, spellin, spellout: find	spelling errors.	spell(1)
spell, spellin,	spelling errors.	spell(1)
spellout: find	spelling errors.	spell(1)
spline: interpolate smooth curve.....	spline(1G)	
split:	split a file into pieces.	split(1)
files. fsplit:	split a multi-routine FORTRAN file into individual	split(1)
frexp, ldexp, modf:	split into mantissa and exponent.	frexp(3)
	split: split a file into pieces.	split(1)
uuclean: uucp	spool directory clean-up.....	uuclean(8C)
lpq:	spool queue examination program.	lpq(1)
lprm: remove jobs from the line printer	spooling queue.....	lprm(1)
printf, fprintf,	sprintf: formatted output conversion.	printf(3S)
exp, log, log10, pow,	sqrt: exponential, logarithm, power, square root.	exp(3M)
log10, pow, sqrt: exponential, logarithm, power,	square root. exp, log,	exp(3M)
cvtumap: convert name trees from	SR8 to SR9 name mapping.	cvtumap(8)
cvtumap: convert name trees from SR8 to	SR9 name mapping.	cvtumap(8)
generator and associated routines. random,	srandom, initstate, setstate: better random number	random(3)
scanf, fscanf,	sscanf: formatted input conversion.....	scanf(3S)
popd: pop shell directory	stack.	csh(1)
pushd: push shell directory	stack.	csh(1)
sigstack: set and/or get signal	stack context.....	sigstack(2)
systype: display version	stamp.....	systype(8)
stdio:	standard buffered input/output package.	intro(3S)
htable: convert NIC	standard format host tables.....	htable(8)
cp /bin/start_sh:	start a Bourne Shell.	start_sh(1)
cp /bin/start_csh:	start a C shell.	start_csh(1)
	stat, lstat, fstat: get file status.	stat(2)
reset: reset the teletype bits to a sensible	state.....	reset(1)
fsync: synchronize a file's in-core	state with that on disk.....	fsync(2)
if: conditional	statement.....	csh(1)
fstab:	static information about filesystems.	fstab(5)
hashstat: print command hashing	statistics.	csh(1)
netstat: show network	status.	netstat(1)
ps: process	status.	ps(1)
stat, lstat, fstat: get file	status.	stat(2)
ferror, feof, clearerr, fileno: stream	status inquiries.....	ferror(3S)
ruptime: show host	status of local machines.	ruptime(1C)
rwhod: system	status server.....	rwhod(8C)
	stdio: standard buffered input/output package.	intro(3S)
	stop: halt a job or process.	csh(1)
halt:	stop the processor.....	halt(8)
pause:	stop until signal.	pause(3C)
subroutines. dbminit, fetch,	store, delete, firstkey, nextkey: database.....	dbm(3X)
strlen, index, rindex: string operations.	strcat, strncat, strcmp, strncmp, strcpy, strncpy,	string(3)
rindex: string operations. strcat, strncat,	strcmp, strncmp, strcpy, strncpy, strlen, index,	string(3)
operations. strcat, strncat, strcmp, strncmp,	strcpy, strncpy, strlen, index, rindex: string.....	string(3)
fclose, fflush: close or flush a	stream.....	fclose(3S)
fopen, freopen, fdopen: open a	stream.....	fopen(3S)

fseek, ftell, rewind: reposition a	stream.....fseek(3S)
getchar, fgetc, getw: get character or word from	stream. getc,getc(3S)
gets, fgets: get a string from a	stream.....gets(3S)
putchar, fputc, putw: put character or word on a	stream. putc,putc(3S)
puts, fputs: put a string on a	stream.....puts(3S)
setbuffer, setlinebuf: assign buffering to a	stream. setbuf,setbuf(3S)
ungetc: push character back into input	stream.....ungetc(3S)
sed:	stream editor.....sed(1)
ferror, feof, clearerr, fileno:	stream status inquiries.....ferror(3S)
rcmd, rresvport, ruserok: routines for returning a	stream to a remote command.rcmd(3X)
rexec: return	stream to a remote command.rexec(3X)
	strfile: fortune(6) database loader.....strfile(6)
gets, fgets: get a	string from a stream.....gets(3S)
puts, fputs: put a	string on a stream.....puts(3S)
bcopy, bcmp, bzero, ffs: bit and byte	string operations.....bstring(3)
strncmp, strcpy, strncpy, strlen, index, rindex:	string operations. strcat, strncat, strcmp,string(3)
extract strings from C programs to implement shared	strings. xstr:.....xstr(1)
file.	strings: find the printable strings in an objectstrings(1)
strings. xstr: extract	strings from C programs to implement sharedxstr(1)
strings: find the printable	strings in an object file.strings(1)
basename:	strip filename affixes.....basename(1)
from an object file.	strip: strip symbol and line number informationstrip(1)
object file. strip:	strip symbol and line number information from an strip(1)
strcat, strncat, strcmp, strncmp, strcpy, strncpy,	strlen, index, rindex: string operations.....string(3)
index, rindex: string operations. strcat,	strncat, strcmp, strncmp, strcpy, strncpy, strlen,string(3)
string operations. strcat, strncat, strcmp,	strncmp, strcpy, strncpy, strlen, index, rindex:string(3)
strcat, strncat, strcmp, strncmp, strcpy,	strlen, index, rindex: string operations.....string(3)
	stty: set terminal options.....stty(1)
document.	style: analyze surface characteristics of a.....style(1)
fetch, store, delete, firstkey, nextkey: database	su: substitute user ID temporarily.....su(1)
su:	subroutines. dbminit,.....dbm(3X)
sum:	substitute user ID temporarily.su(1)
	sum and count blocks in a file.sum(1)
	sum: sum and count blocks in a file.sum(1)
du:	summarize disk usage.du(1)
	sup: set UNIX-style protection.sup(8)
sync: update	super-block.sync(2)
sync: update the	super-block.sync(8)
update: update the	super-block periodically.....update(8)
inetd: Internet	superdaemon.....inetd(8C)
suspend: suspend a shell, resuming its	superior.csh(1)
style: analyze	surface characteristics of a document.style(1)
suspend:	suspend a shell, resuming its superior.csh(1)
sleep:	suspend execution for an interval.....sleep(1)
sleep:	suspend execution for interval.....sleep(3)
	suspend: suspend a shell, resuming its superior.csh(1)
swab:	swab: swap bytes.....swab(3)
	swap bytes.swab(3)
breaksw: exit from	swapul: rearrange underliningswapul(8)
	switch.....csh(1)

case: selector in	switch.....	csh(1)
default: catchall clause in	switch.....	csh(1)
endsw: terminate	switch.....	csh(1)
	switch: multi-way command branch.	csh(1)
file. strip: strip	symbol and line number information from an objectstrip(1)	
readlink: read value of a	symbolic link.	readlink(2)
symlink: make	symbolic link to a file.....	symlink(2)
	symlink: make symbolic link to a file.	symlink(2)
	sync: update super-block.	sync(2)
	sync: update the super-block.	sync(8)
disk. fsync:	synchronize a file's in-core state with that on.....	fsync(2)
select:	synchronous I/O multiplexing.	select(2)
csh: a shell (command interpreter) with C-like	syntax.	csh(1)
perror,	sys_errlist, sys_ner: system error messages.	perror(3)
	syslog: log systems messages.	syslog(8)
perror, sys_errlist,	sys_ner: system error messages.	perror(3)
psignal,	sys_siglist: system signal messages.....	psignal(3)
tip, cu: connect to a remote	system.	cu(1C)
hostid: set or print identifier of current host	system.	hostid(1)
hostname: set or print name of current host	system.	hostname(1)
mount, umount: mount or remove file	system.	mount(2)
mount, umount: mount and dismount file	system.	mount(8)
tip, cu: connect to a remote	system.	tip(1C)
users: compact list of users who are on the	system.	users(1)
who: who is on the	system.	who(1)
syslog: log	systems messages.	syslog(8)
	systype: display version stamp.	systype(8)
rehash: recompute command hash	table.	csh(1)
unhash: discard command hash	table.	csh(1)
mtab: mounted file system	table.	mtab(5)
getdtablesize: get descriptor	table size.....	getdtablesize(2)
htable: convert NIC standard format host	tables.....	htable(8)
route: manually manipulate the routing	tables.....	route(8C)
tbl: format	tables for nroff or troff.	tbl(1)
gettable: get NIC format host	tables from a host.....	gettable(8C)
tabs: set terminal	tabs.....	tabs(1)
	tabs: set terminal tabs.	tabs(1)
expand, unexpand: expand	tabs to spaces and vice versa.....	expand(1)
ctags: create a	tags file.	ctags(1)
	tail: deliver the last part of a file.	tail(1)
	talk: talk to another user.....	talk(1)
talk:	talk to another user.	talk(1)
talkd: server for	talk(1) program.....	talkd(8C)
	talkd: server for talk(1) program.	talkd(8C)
functions. sin, cos,	tan, asin, acos, atan, atan2: trigonometric.....	sin(3M)
sinh, cosh,	tanh: hyperbolic functions.	sinh(3M)
tar:	tape (and general purpose) archiver.	tar(1)
tar:	tape archive file format.....	tar(5)
mtio:	tape device files.....	mtio(4)
mt: magnetic	tape manipulating program.	mt(1)

	tar: tape (and general purpose) archiver.tar(1)
	tar: tape archive file format.tar(5)
deroff: remove nroff, troff,	tbl, and eqn constructs.deroff(1)
	tbl: format tables for nroff or troff.tbl(1)
	tcp: Internet Transmission Control Protocol.tcp(4P)
	tee: pipe fitting.tee(1)
reset: reset the	teletype bits to a sensible state.reset(1)
operations. opendir, readdir,	tell, seekdir, rewinddir, closedir: directory.directory(3)
telnet: user interface to the	TELNET protocol.telnet(1C)
telnetd: DARPA	TELNET protocol servertelnetd(8C)
	telnet: user interface to the TELNET protocol.telnet(1C)
	telnetd: DARPA TELNET protocol server.telnetd(8C)
su: substitute user ID	temporarily.su(1)
	term: conventional names for terminals.term(7)
	termcap: terminal capability database.termcap(5)
ttyname, isatty: find name of a	terminal.ttyname(3)
worms: animate worms on a display	terminal.worms(6)
termcap:	terminal capability database.termcap(5)
pty: pseudo	terminal driver.pty(4)
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs:	terminal independent operation routines.termcap(3X)
tty: general	terminal interface.tty(4)
tty: get	terminal name.tty(1)
stty: set	terminal options.stty(1)
clear: clear	terminal screen.clear(1)
script: make typescript of a	terminal session.script(1)
tabs: set	terminal tabs.tabs(1)
tset:	terminal-dependent initialization.tset(1)
term: conventional names for	terminals.term(7)
wait, wait3: wait for process to	terminate.wait(2)
_exit:	terminate a process.exit(2)
output. exit:	terminate a process after flushing any pending.exit(3)
kill:	terminate a specified process.kill(1)
endif:	terminate conditional.csh(1)
end:	terminate loop.csh(1)
endsw:	terminate switch.csh(1)
	test: condition command.test(1)
ed:	text editor.ed(1)
ex, edit:	text editor.ex(1)
eqn: format mathematical	text for troff.eqn(1)
fmt: simple	text formatter.fmt(1)
nroff:	text formatting.nroff(1)
troff:	text formatting and typesetting.troff(1)
ms:	text formatting macros.ms(7)
	tftpd: DARPA Trivial File Transfer Protocol server.tftpd(8C)
terminal independent operation routines.	tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs:termcap(3X)
independent operation routines. tgetent, tgetnum,	tgetflag, tgetstr, tgoto, tputs: terminaltermcap(3X)
independent operation routines. tgetent,	tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal.termcap(3X)
operation routines. tgetent, tgetnum, tgetflag,	tgetstr, tgoto, tputs: terminal independent.termcap(3X)
routines. tgetent, tgetnum, tgetflag, tgetstr,	tgoto, tputs: terminal independent operationtermcap(3X)
ccat: compress and uncompress files, and then cat	them. compact, uncompact,compact(1)

uncompact, ccat: compress and uncompress files, and diction, explain: print wordy sentences; diff3: at: execute commands at a later gettimeofday, settimeofday: get/set date and time: time: rc: boot	then cat them. compact.....compact(1) thesaurus for diction.....diction(1) three-way differential file comparison.diff3(1) time.at(1) time.gettimeofday(2) time a command.....time(1) time command.....csh(1) time shell script.....rc(8) time: time a command.....time(1) time: time command.....csh(1) time to ASCII. ctime, localtime,.....ctime(3) timer.....gettimer(2) times.....times(3C) times.....utimes(2) times: get process times.....times(3C) timezone: convert date and time to ASCII.....ctime(3) tip, cu: connect to a remote system.....cu(1C) tip, cu: connect to a remote system.....tip(1C) tsort: topological sort.....tsort(1) touch: update date last modified of a file.....touch(1) tputs: terminal independent operation routines.....termcap(3X) tr: translate characters.....tr(1) ptrace: process goto: command ftp: file ftpd: DARPA Internet File tftpd: DARPA Trivial File tr: tcp: Internet uuencode,uudecode: encode/decode a binary file for cvtumap: convert name trek: sin, cos, tan, asin, acos, atan, atan2: tftpd: DARPA eqn: format mathematical text for tbl: format tables for nroff or deroff: remove nroff, false, truncate: false, true: provide true, false: provide crpty: create psuedo	trace.....ptrace(2) transfer.....csh(1) transfer program.....ftp(1C) Transfer Protocol server.....ftpd(8C) Transfer Protocol server.....tftpd(8C) translate characters.....tr(1) Transmission Control Protocol.....tcp(4P) transmission via mail.....uuencode(1C) trees from SR8 to SR9 name mapping.....cvtumap(8) trek: trekkie game.....trek(6) trekkie game.....trek(6) trigonometric functions.....sin(3M) Trivial File Transfer Protocol server.....tftpd(8C) troff.....eqn(1) troff.....tbl(1) troff, tbl, and eqn constructs.....deroff(1) troff: text formatting and typesetting.....troff(1) true, false: provide truth values.....true(1) true: provide truth values.....false(1) truncate a file to a specified length.....truncate(2) truncate: truncate a file to a specified length.....truncate(2) truth values.....false(1) truth values.....true(1) tset: terminal-dependent initialization.....tset(1) tsort: topological sort.....tsort(1) tty device entries.....crpty(8) tty: general terminal interface.....tty(4) tty: get terminal name.....tty(1)
--	---	--

	ttyname, isatty: find name of a terminal.	ttyname(3)
file: determine file	type.	file(1)
types: primitive system data	types.	types(5)
	types: primitive system data types.	types(5)
script: make	typescript of a terminal session.	script(1)
troff: text formatting and	typesetting.	troff(1)
	udp: Internet User Datagram Protocol.	udp(4P)
	ul: do underlining.	ul(1)
	umask: change or display file creation mask.	csh(1)
	umask: set/get file creation mask.	umask(2)
mount,	umount: mount and dismount file system.	mount(8)
mount,	umount: mount or remove file system.	mount(2)
	unalias: remove aliases.	csh(1)
then cat them. compact,	uncompact, ccat: compress and uncompress files, and	compact(1)
compact, uncompact, ccat: compress and	uncompress files, and then cat them.	compact(1)
swapul: rearrange	underlining.	swapul(8)
ul: do	underlining.	ul(1)
unget:	undo a previous get of an SCCS file.	unget(1)
expand,	unexpand: expand tabs to spaces and vice versa.	expand(1)
	unget: undo a previous get of an SCCS file.	unget(1)
	ungetc: push character back into input stream.	ungetc(3S)
	unhash: discard command hash table.	csh(1)
	uniq: report repeated lines in a file.	uniq(1)
mktemp: make a	unique filename.	mktemp(3)
gethostid, sethostid: get/set	unique identifier of current host.	gethostid(2)
	units: conversion program.	units(1)
uucp, uuname, uulog: UNIX to	UNIX copy.	uucp(1C)
uucp, uuname, uulog:	UNIX to UNIX copy.	uucp(1C)
sup: set	UNIX-style protection.	sup(8)
uux:	UNIX-to-UNIX command execution.	uux(1C)
	unlimit: remove resource limitations.	csh(1)
rm, rmdir: remove	(unlink) directories or files.	rm(1)
	unlink: remove directory entry.	unlink(2)
	unset: discard shell variables.	csh(1)
	unsetenv: remove environment variables.	csh(1)
uptime: show how long a node has been	up.	uptime(1)
update_slave:	update auxiliary system administrator's nodes.	update_slave(8)
touch:	update date last modified of a file.	touch(1)
sync:	update super-block.	sync(2)
sync:	update the super-block.	sync(8)
update:	update the super-block periodically.	update(8)
	update: update the super-block periodically.	update(8)
administrator's nodes.	update_slave: update auxiliary system.	update_slave(8)
	uptime: show how long a node has been up.	uptime(1)
du: summarize disk	usage.	du(1)
checkeq: check files that	use eqn(1) or neqn(1).	checkeq(1)
intro: miscellaneous	useful information pages.	intro(7)
login: login new	user.	csh(1)
talk: talk to another	user.	talk(1)
write: write to another	user.	write(1)

setuid setreuid setruid setgid setegid setrgid: set	user and group IDnet(3n)
seteuid, setruid, setgid, setegid, setrgid: set	user and group ID. setuid,setuid(3)
udp: Internet	User Datagram Protocol.....udp(4P)
setreuid: set real and effective	user ID.setreuid(2)
whoami: print effective current	user ID.whoami(1)
su: substitute	user ID temporarily.....su(1)
getuid, geteuid: get	user identity.getuid(2)
telnet:	user interface to the TELNET protocol.telnet(1C)
binmail: send or receive mail among	users.binmail(1)
	users: compact list of users who are on the system.users(1)
wall: write to all	users on a node.wall(1)
users: compact list of	users who are on the system.....users(1)
getrusage: get information about resource	utilization.getrusage(2)
	utimes: set file times.utimes(2)
	uuclean: uucp spool directory clean-upuuclean(8C)
mail: handle remote mail received via	uucp.rmail(1)
uuclean:	uucp spool directory clean-upuuclean(8C)
uusnap: show snapshot of the	UUCP systemuusnap(8C)
	uucp, uuname, uulog: UNIX to UNIX copy.....uucp(1C)
uuencode: format of an encoded	uuencode file.uuencode(5)
	uuencode: format of an encoded uuencode file.uuencode(5)
transmission via mail.	uuencode,uudecode: encode/decode a binary file foruuencode(1C)
uucp, uuname,	uulog: UNIX to UNIX copy.....uucp(1C)
uucp,	uuname, uulog: UNIX to UNIX copy.....uucp(1C)
	uusend: send a file to a remote host.uusend(1C)
	uusnap: show snapshot of the UUCP systemuusnap(8C)
	uux: UNIX-to-UNIX command execution.....uux(1C)
	val: validate SCCS file.....val(1)
val:	validate SCCS file.val(1)
	valloc: aligned memory allocator.....valloc(3)
abs: integer absolute	value.....abs(3)
fabs, floor, ceil: absolute	value, floor, ceiling functions.floor(3M)
readlink: read	value of a symbolic link.readlink(2)
getenv: get the	value of an environment variable.getenv(3)
getitimer, setitimer: get/set	value of interval timer.getitimer(2)
set: change	value of shell variable.....csh(1)
false, true: provide truth	values.false(1)
true, false: provide truth	values.true(1)
htonl, htons, ntohl, ntohs: convert	values between host and network byte order.....byteorder(3n)
vfprintf, vsprintf: print formatted output of a	varargs argument list. vprintf,vprintf(3S)
	varargs: variable argument list.varargs(3)
set: change value of shell	variable.csh(1)
getenv: get the value of an environment	variable.getenv(3)
varargs:	variable argument list.varargs(3)
setenv: set	variable in environment.csh(1)
@: arithmetic on shell	variables.....csh(1)
unset: discard shell	variables.....csh(1)
unsetenv: remove environment	variables.....csh(1)
environ: environment	variables.....environ(7)
	ver: change the version of Shell commands.....ver(8)

assert: program	verification.....	assert(3X)
lint: a C program	verifier.....	lint(1)
expand, unexpand: expand tabs to spaces and vice versa.....	versa.....	expand(1)
get: get a	version of an SCCS file.....	get(1)
ver: change the	version of Shell commands	ver(8)
hangman: Computer	version of the hangman game.....	hangman(6)
systype: display	version stamp.....	systype(8)
sccsdiff: compare two	versions of an SCCS file.....	sccsdiff(1)
varargs argument list. vprintf, on ex.	vfork: spawn a new process in a more efficient way.....	vfork(2)
encode/decode a binary file for transmission	vfprintf, vsprintf: print formatted output of a.....	vprintf(3S)
rmail: handle remote mail received	vi: screen-oriented (visual) display editor based	vi(1)
expand, unexpand: expand tabs to spaces and	via mail. uuencode,uudecode:	uuencode(1C)
more, page: file perusal filter for CRT	via uucp.....	rmail(1)
more, page: file perusal filter for CRT	vice versa.....	expand(1)
vi: screen-oriented	viewing.....	more(1)
of a varargs argument list.	viewing.....	page(1)
argument list. vprintf, vfprintf,	(visual) display editor based on ex.....	vi(1)
wait:	vprintf, fprintf, vsprintf: print formatted output	vprintf(3S)
sigpause: atomically release blocked signals and	vsprintf: print formatted output of a varargs	vprintf(3S)
wait, wait3:	wait: await completion of process.....	wait(1)
wait,	wait for background processes to complete.....	csh(1)
vfork: spawn a new process in a more efficient	wait for interrupt.....	sigpause(2)
whatis: describe	wait for process to terminate.....	wait(2)
leave: remind you	wait: wait for background processes to complete.....	csh(1)
paths.	wait, wait3: wait for process to terminate.....	wait(2)
break: exit	wait3: wait for process to terminate.....	wait(2)
users: compact list of users	wall: write to all users on a node.....	wall(1)
from:	way.....	vfork(2)
who:	wc: word count.....	wc(1)
rwho:	what a command is.....	whatis(1)
fold: fold long lines for finite	what: identify SCCS files.....	what(1)
wc:	whatis: describe what a command is.....	whatis(1)
getc, getchar, fgetc, getw: get character or	when you have to leave.....	leave(1)
putc, putchar, fputc, putw: put character or	whereis: locate binary and/or manual for program.....	whereis(1)
diction, explain: print	which: locate a program file, including aliases and which.....	which(1)
cd: change	while: repeat commands conditionally.....	csh(1)
chdir: change current	while/foreach loop.....	csh(1)
	who are on the system.....	users(1)
	who is my mail from?.....	from(1)
	who: who is on the system.....	who(1)
	who: who is on the system.....	who(1)
	whoami: print effective current user ID.....	whoami(1)
	who's logged in on local machines.....	rwho(1C)
	width output device.....	fold(1)
	word count.....	wc(1)
	word from stream.....	getc(3S)
	word on a stream.....	putc(3S)
	wordy sentences; thesaurus for diction.....	diction(1)
	working directory.....	cd(1)
	working directory.....	chdir(2)

pwd:	working directory name.	pwd(1)
getwd: get current	working directory pathname.	getwd(3)
worm: Play the growing	worm game.	worm(6)
	worm: Play the growing worm game.	worm(6)
	worms: animate worms on a display terminal.	worms(6)
worms: animate	worms on a display terminal.	worms(6)
write, writev:	write on a file.	write(2)
wall:	write to all users on a node.	wall(1)
write:	write to another user.	write(1)
	write: write to another user.	write(1)
	write, writev: write on a file.	write(2)
writed: daemon for	write(1) program.	writed(8C)
	writed: daemon for write(1) program.	writed(8C)
write,	writev: write on a file.	write(2)
open: open a file for reading or	writing, or create a new file.	open(2)
shared strings.	xstr: extract strings from C programs to implement xstr(1)	
j0, j1, jn,	y0, y1, yn: Bessel functions.	j0(3M)
j0, j1, jn, y0,	y1, yn: Bessel functions.	j0(3M)
eyacc: modified	yacc allowing much improved error recovery.	eyacc(1)
	yacc: yet another compiler-compiler.	yacc(1)
	yes: be repetitively affirmative.	yes(1)
j0, j1, jn, y0, y1,	yn: Bessel functions.	j0(3M)

READER'S RESPONSE FORM

We use readers' comments in revising and improving our documents.

DOMAIN/IX Command Reference for System V, Order No. 005798, Revision 01

Date of Publication: December 1986

What is the best feature of this manual?

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible.)

What type of user are you?

- ☐ Systems programmer; language
- ☐ Applications programmer; language
- ☐ Manager/Professional
- ☐ Administrative/Support Personnel
- ☐ Student programmer
- ☐ User with little programming experience
- ☐ Other

How often do you use the DOMAIN system?

Nature of your work on the DOMAIN system:

expand;

l l l.

Your name

Date

Organization

Street Address

City State Zip/Country

No postage necessary if mailed in the U.S. Fold on dotted lines (see reverse side), tape, and mail.

cut or fold along dotted line

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

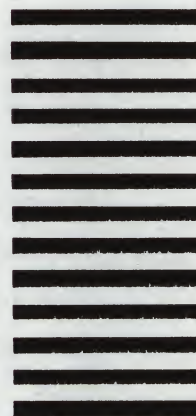
FIRST CLASS

PERMIT NO. 78

CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824



FOLD

READER'S RESPONSE FORM

We use readers' comments in revising and improving our documents.

DOMAIN/IX Command Reference for System V, Order No. 005798, Revision 01
Date of Publication: December 1986

What is the best feature of this manual?

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible.)

What type of user are you?

- ☐ Systems programmer; language
- ☐ Applications programmer; language
- ☐ Manager/Professional
- ☐ Administrative/Support Personnel
- ☐ Student programmer
- ☐ User with little programming experience
- ☐ Other

How often do you use the DOMAIN system?

Nature of your work on the DOMAIN system:

expand;

111.

Your name

Date

Organization

Street Address

City State Zip/Country

No postage necessary if mailed in the U.S. Fold on dotted lines (see reverse side), tape, and mail.

cut or fold along dotted line

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS

PERMIT NO. 78

CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824



FOLD

Instruction Sheet

Insert Tabbed Divider Page:

Section 1
Section 6
Permuted Index

Before Page:

1-i
6-i
A-1

